de tekstmachine

dr. M. BOOT
drs. H. KOPPELAAR

ACADEMIC SERVICE

de tekstmachine

de tekstmachine

dr. M. BOOT
drs. H. KOPPELAAR

ACADEMIC SERVICE

Uitgegeven door: Academic Service
Postbus 96996
2509 JJ Den Haag
Zetwerk: mevr. I. Geerling-Engelbarts
Illustraties: Eddy Both
Ontwerp omslag: JAM Gauw
Druk: Krips Repro, Meppel
Bindwerk: Meeuwis, Amsterdam
ISBN: 90 6233 071 1

INHOUD

INLEIDING

Dit boek handelt over de instrumenten die de machine in staat stellen op zinnige wijze om te gaan met teksten in natuurlijke taal. Voor we echter ingaan op concrete instrumenten (computertalen en software packages) willen we eerst aandacht besteden aan de vraag of het wel zinvol is de computer te willen programmeren voor dit soort taken. Is het immers niet een algemeen aanvaard standpunt dat computers te enen male incompetent zijn voor de behandeling van tekstmateriaal? Nemen we bijvoorbeeld onze dagelijkse krant, waarvan de computer de woordafbrekingsperikelen dient op te lossen. Gegarandeerd dat we fouten vinden als: 'zoont-jes', 'cont-role', 'sterks-te', 'inkomensachteruitgang' (allemaal uit de NRC d.d. 18 en 19 maart 1980). Wel, redeneert men dan, wanneer een zo eenvoudig probleem als dat der woordafbreking –een kind leert het immers al correct– niet kan worden opgelost, wat kan men dan verder doen aan tekstprocessing? De mens gaat immers best met teksten om en men kan nauwelijks zeggen dat het zo moeilijk voor hem is; na enige training kan een mens zelfs de meest onmogelijke gedichten leren begrijpen en wat nog erger is: hij kan ze nog zelf maken ook. Wat moet een dorre cijferaar als een computer dan beginnen met al die taken die toch erg intuïtief van aard zijn? Dat kan toch helemaal niets opleveren en mocht er dan iemand zijn die beweert een computerprogramma te hebben gemaakt dat teksten kan vertalen, dan moet er toch wel sprake zijn van bedrog?

Kortom, er lijken allerlei argumenten te bestaan die onderstrepen dat de computer te enen male onbekwaam lijkt om op andere dan onmondige wijze met taal en tekst om te gaan.

Het valt niet te ontkennen dat de meerderheid van de computerprogramma's, die tot nu toe voor textprocessing werden gemaakt, zich niet erg indrukwekkend van de gestelde taak gekweten hebben. Ook kan tegenwoordig nauwelijks meer verborgen blijven dat onder textprocessing zelden méér wordt verstaan dan het maken van bijvoorbeeld ongedefinieerde woordenlijsten, waarbij wel netjes wordt geteld hoeveel maal bijvoorbeeld het Nederlandse woordje 'was' voorkomt, maar waarbij niet wordt onderscheiden of het gaat om 'was' uit: de was hangt buiten, of 'was' uit: het was mooi weer. In toepassingen voor het kantoor onderscheidt men textprocessing en wordprocessing. Wordpro-

cessing is het opleveren van brieven/contracten die grotendeels standaard zijn, tezamen met de optie om daarin wijzigingen te kunnen aanbrengen.

Het zal niemand enige moeite kosten zelfs in het Nederlandse taalgebied een stuk of vijf projecten op te snorren waarin men bezig is met het ontwerpen van computerprogramma's die weer eens bovengenoemd soort woordenlijsten gaan maken. Meestal verstaat men onder textprocessing helaas niet veel meer dan het uitvoeren van sorteerprogramma's. Deze sorteerprogramma's schrijft men dan soms uit onwetendheid in een programmeertaal die slecht is toegerust voor het vinden van woorden ('strings'). Op die manier ontstaan veelal schijnproblemen. Deze 'problemen' gaat men dan in den brede behandelen, bestuderen en oplossen; 'problemen' die niet worden veroorzaakt door de probleemstelling zelf (het vinden van een woord in een tekst), maar door de kreupelheid van het gebruikte instrument, dat wil zeggen de programmeertaal.

Men kan ook weer zelf het wiel gaan uitvinden, dat is in dit geval een ongeschikte programmeertaal aanpassen voor tekstverwerking. Wij willen met dit boek niet weer het wiel uitvinden. De geschiedenis van vakken als computerlinguistiek en kunstmatige intelligentie heeft wel degelijk instrumenten opgeleverd voor het probleemveld dat we willen betreden. Dat probleemveld kan men samenvatten in de vraag: welke instrumenten gebruikt men om op handige wijze computerprogramma's te schrijven die de machine in staat stellen op zinnige wijze om te gaan met teksten in natuurlijke taal?

KENMERKEN VAN HET GEBIED 'TEKST EN COMPUTER'

In de informatica leeft de droom van de ene algemene, uiterst mooie programmeertaal, die alles zou moeten kunnen oplossen als hij maar behoorlijk ontworpen was. Daarbij gaat men ervan uit dat op getallen en rekenen gerichte (zogenaamde numerieke) en op tekstverwerking gerichte (zogenaamde niet-numerieke) toepassingen t h e o r e t i s c h niet zó verschillen. Het is in deze opvatting immers dezelfde tweetallige ('lineaire', digitale) machine die de problemen diep in zijn 'brein' moet oplossen. In de praktijk, de natuurlijke wildbaan, is het echter nog niet gelukt een dergelijke programmeertaal te fokken. De Amerikaanse programmeertaal die als 'language for all seasons' werd betiteld, namelijk de algorithmische taal PL/I, is zo omvangrijk als een encyclopedie geworden. De als in de hemel geconcipieerde taal ALGOL68 is een beknopte, daardoor zeer arbeidsintensieve en dus uiterst dure grap gebleken. Een grap, die zeer gebruikersonvriendelijk is voor textprocessing. PASCAL is weliswaar een prettige, beknopte taal, maar is ongeschikt voor textprocessing, tenzij men weer het wiel wil uitvinden. Ook FORTRAN wordt alsmaar weer geadapteerd

aan textprocessing, maar zonder echt resultaat.

Bij al deze numeriek georienteerde talen blijft voor tekstbehandeling het probleem, dat men zich steeds bezig moet houden met problemen die niet bij het gebied zelf horen. Het zijn problemen die samenhangen met de interne structuur van de computer en met de interne structuur van teksten. Kortom coderingsproblemen. Het minst lastig hierbij is zonder twijfel PL/I. Tegen de tijd dat men de problemen met behulp van een numerieke programmeertaal heeft opgelost die bij de computer horen, zijn de programma's meestal zo gecompliceerd geworden, dat niemand er meer echt wijs uit kan worden; of is de subsidie op.

Wat programmeertalen betreft is het zonder meer duidelijk dat er voor werkelijke textprocessing niet é é n enkele programmeertaal bestaat die overal tegelijk goed voor is. Vroeger verstond men in Europa onder een bekwaam programmeur iemand die de meeste programmeerproblemen met behulp van ALGOL60 kon oplossen. Hij moest vooral in getallen en rijen getallen kunnen denken. Nu (met name op het gebied van taal en computer) gaat dat niet meer op. Textprocessing is geen getallen- (numeriek) werk. Men moet het dan ook niet op numerieke wijze proberen op te lossen. De oplossing van het textprocessingsprobleem krijgt men het best in de vingers door gebruik te maken van de v e r s c h i l l e n d e niet-numerieke instrumenten die daarvoor (bijvoorbeeld op het MIT) zijn ontwikkeld. Behalve door dit algemene standpunt wordt de beslissing om als tekstverwerker de beginselen van meer dan één programmeertaal te leren ook bevestigd door de praktijk. Belangrijke programmatuur die nog in ontwikkeling is, wordt nog steeds in diverse programmeertalen geschreven. De aard van het op te lossen probleem brengt met zich mee dat het ontwikkelde computerprogramma dikwijls het enige formalisme is dat een adequate beschrijving van de probleemstelling inhoudt. Zodoende kan ook de theoretisch ingestelde onderzoeker van 'taal en textprocessing' zich niet meer permitteren de terzake doende programmeertalen niet te kunnen lezen.

Inderdaad is het zo, dat een belangrijke nieuwe ontwikkeling op het gebied der menselijke mogelijkheden op gang begint te komen. Zijn taal- en letterkunde van oudsher statisch en structureel georiënteerd, nu begint men zich daar meer bezig te houden met de vraag hoe de mens taal verwerkt. Het beste instrument dat de onderzoeker vandaag de dag heeft om taalverwerkingsprocessen te bestuderen is de computer. De computer is immers een instrument dat processen uitvoert. Een dergelijke instrument is de taal- en letterkundige echter van huis uit vreemd. Wat men met textprocessing zou m o e t e n doen zijn verwerkingsprocessen in verband met de symbolische informatie die taal nu eenmaal is. Daarom kan men zich voor een dergelijke verwerking niet blijven beperken tot het manipuleren van woorden ('lexicale ingang'). Men dient voor werkelijke textprocessing ook in staat te zijn het menselijk taalvermogen enigszins na te bootsen. Moet iemand bijvoorbeeld een bewijs afleggen dat hij een tekst begrijpt, dan zal hij in principe drie soorten handelingen moeten kunnen verrichten. Hij moet

kunnen denken (conclusies trekken bijvoorbeeld). Teksten begrijpen heeft dus een denk- of 'cognitief' aspect. Verder zal hij woorden moeten kunnen herkennen en opzoeken in zijn 'interne' woordenboek. Dit kan men aanduiden met het 'zoek-' of 'geheugen-'aspect. Hieronder vallen ook handelingen als het herkennen van structuren (van zinnen bijvoorbeeld). Tot slot zal de gebruiker van taal ook moeten doen blijken dat hij de tekst heeft begrepen. Hij zal er iets over willen zeggen. Dit kan men aanduiden met het productief of generatief aspect.

Wij onderscheiden dus drie aspecten van text-/taalprocessing: een cognitief aspect, een zoekaspect (geheugenaspect) en een expressief (generatief) aspect. Dit laatste, het generatief aspect, zou men ook het taalkundig aspect kunnen noemen, tenminste voor zover men zich bevindt op het niveau van de zin. Op andere wijze zou men nu de aspecten aldus kunnen formuleren: textprocessing heeft een talig aspect (handling linguistic information), een tekstaspect (handling textual information) en een psychologisch aspect (handling cognitive information en ook memory processes). Juist deze inhoudelijke complexiteit van het hele gebied van menselijke informatieverwerking door taal heeft veroorzaakt dat er de facto verschillende instrumenten zijn ontwikkeld. Om het gebied overzichtelijk te houden is dat juist een groot voordeel. Aparte gebieden van processing houdt men op die manier uit elkaar. Men komt niet in de verleiding bijvoorbeeld met een getallenmachine dingen te willen oplossen die niet getalsmatig definieerbaar zijn. Het is toch te gek dat men bijvoorbeeld om een woordafbreekprogramma te kunnen maken eerst een code moet verzinnen die letters in getallen omzet. Toch hebben alle computerprogramma's die tegenwoordig hiervoor in Nederland gebruikt worden juist dit kenmerk. Eenzelfde opmerking zou men kunnen maken als het gaat om de programma's die zinnen ontleden. Ook hier vindt men als grondpatroon voor het maken van computerprogramma's een onderliggend concept van analyse, namelijk de contextvrije hiërarchisch georganiseerde (boomstructuur) grammatica, waarvan aantoonbaar is dat die niet zo erg veel te maken heeft met menselijke tekstverwerking (information processing). De werkelijke reden daarvoor is natuurlijk dat computertalen zelf gemaakt worden aan de hand van dat soort grammatica's. Het besef dat programmeertalen wellicht helemaal geen aantoonbaar verband hebben met menselijke taal wordt daarbij voor het gemak over het hoofd gezien. De grondpremisse dat intelligente tekstverwerking zich niet hoeft bezig te houden met processen in echte of 'natuurlijke' taal, is voor het gebied van textprocessing natuurlijk te gek. Ook het idee dat een grammatica een hoeveelheid taalkundige informatie z o n d e r b e t e k e n i s zou bevatten, los van andere linguïstische componenten zoals 'semantiek' of 'pragmatiek', is alleen vol te houden wanneer men een laboratoriumsituatie ontwikkelt waarin men zich niet bekommert om praktische informatieverwerking. Een erg zinvol gebruik van de computer houdt het o.i. niet in. Van de andere kant is het zo dat men nog nooit een dergelijk machtig apparaat heeft gehad om echte taalprocessen na te bootsen. Het is al zo dikwijls gezegd: de computer dwingt tot

inzichten en vraagstellingen die zonder het apparaat niet aan de orde zouden kunnen komen.

Eén van die vragen is al in het vorenstaande aangegeven, de vraag namelijk naar de realiteitswaarde van taalkundige theorieën.

Een andere waardevolle bijdrage die de computer al heeft geleverd aan menselijke informatieverwerking is het besef, dat wat de mens allemaal zomaar losjesweg doet, een zeer complex mechanisme vooronderstelt. De mens is er echter aan gewend dat soort dingen zomaar te doen. Hij heeft dus niet in de gaten dat het misschien heel ingewikkeld is en komt daardoor met verkeerde theoretische modellen aan. Zo beseft toch eigenlijk niemand hoe verbazingwekkend het wel is, dat een mens een tekst leest, er een vraag over krijgt en die vraag nog kan beantwoorden ook. Wat men waarneemt is niet meer dan het volgende: iemand hoort een vraag en vrijwel direct geeft hij een antwoord. Een proces dat meestal zonder inspanning en vooral onbetwist verloopt. Voor de mens gaat het dus om een simpele activiteit. Maar is die activiteit ook zo simpel? Als we bij het vooroordeel blijven dat dit soort zaken zo simpel is, zullen we dan ooit in staat zijn de processen die eraan ten grondslag liggen serieus te gaan bestuderen? Anders gezegd: zullen we dan ooit een reëel model van menselijke activiteit in dit opzicht krijgen? Op een eenvoudiger voorbeeld toegespitst: is het wel zo eenvoudig woorden af te breken? Weten we wel zo precies welke informatie nodig is om die taak te kunnen volbrengen? Wanneer men de literatuur erop naleest zal het geen moeite kosten deze vraag met een hardgrondig neen te beantwoorden.

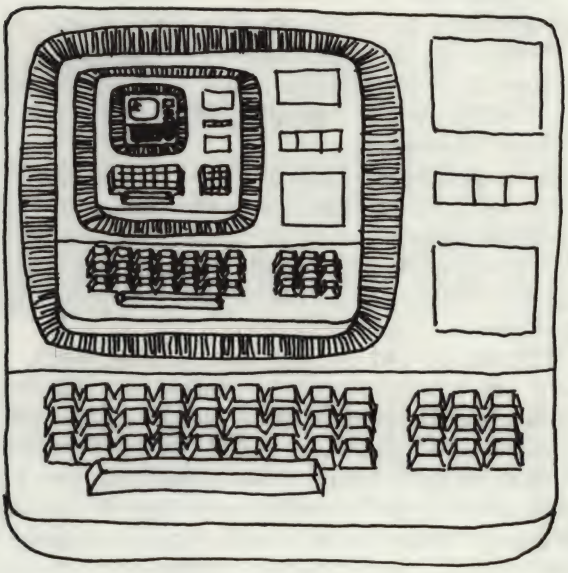
Dit soort problemen wordt in de automatiseringswereld altijd bestudeerd vanuit de mogelijkheden van een bepaalde programmeertaal, nooit vanuit het standpunt van: 'hoe leg ik precies de benodigde informatie vast?' Laat staan vanuit het standpunt: 'hoe en met welke instrumenten organiseer ik die informatie?' Een ander voorbeeld van een wat eenvoudiger soort dan de vraag: 'hoe beantwoordt de mens vragen in taal?' is de vraag: 'hoe zou de mens komen tot woordklastoekenning?'. Dit probleem wordt in feite al voor onoplosbaar gehouden, terwijl wij toch al op de lagere school het onderscheid leren maken tussen werkwoorden, zelfstandige naamwoorden, enz. Pakt men het probleem echter aan vanuit de vraag naar informatie die men verwerkt bij woordklastoekenning, dus niet vanuit de vraag hoe men structuren benoemt, dan blijkt het probleem plotseling wel oplosbaar te zijn. Er kon dan ook een computerprogramma worden ontwikkeld dat deze detailtaak oplost. Woordklastoekenning is echter een probleem dat voor zeer veel soorten van textprocessing een basisvoorwaarde is. Het betekent ontleding op 'categoriaal' niveau. Iedere zinsontleder bouwt voort op de resultaten van dit niveau. Allerlei soorten tekstonderzoek gebruiken de categoriale grammatica als meestal enige taalkundige beschrijving van het corpus. Het probleem van de 'was' dat hiervoor werd aangeduid is dus eigenlijk opgelost wanneer men maar eenmaal bereid is de computer te gebruiken zoals hij voor taal en tekst op zijn sterkst is. Namelijk als de machine waarmee men complexe proble-

men procedureel kan oplossen. De tekstmachine is dus geen abstracte automaat ten dienste van de theoretische taalkunde of de theoretische informatica. De theoretische taalkunde heeft genoeg aan de formalismen die nu ter beschikking staan. De theoretische informatica heeft nauwelijks interesse in de oplossing van concrete problemen en acht zich doorgaans verheven boven het schrijven van computerprogramma's. De tekstmachine lost echter wel concrete problemen op en formaliseert ze tot een gedetailleerde graad van concreetheid, waartoe de 'theorie' niet in staat is en ook niet in staat wil zijn.

Aan het begin van deze inleiding werd het algemeen vooroordeel aan de orde gesteld, dat de computer bij uitstek incompetent zou zijn voor het oplossen van taalkundige problemen. Het zal duidelijk zijn dat deze stelling berust op een grove misvatting. De wetenschap heeft in haar hele geschiedenis nog geen apparaat tot haar beschikking gehad dat juist bij uitstek zo competent is voor de formalisering en de uitvoering van het praktisch nog onbetreden gebied van menselijke informatieverwerking. Het vooroordeel dat de computer incompetent zou zijn, berust op een verkeerde definitie van de computer. Deze verkeerde definitie wordt veroorzaakt door de getallenmaniakkerij van degenen die zich tot nu toe hebben bezig gehouden met problemen op dit gebied. Zij die tot nu toe programmatuur hebben ontworpen voor bijvoorbeeld woordafbreking, werden niet gehinderd door een uitgebreide kennis van Letteren en Psycholinguïstiek, laat staan van Kunstmatige Intelligentie en Patroonherkenning. Degenen die wel door dergelijke kennis worden gehinderd, kunnen zich dan ook niet herkennen in de als wetenschappelijk aangedragen en gepresenteerde modellen voor woordafbreking. Hetzelfde geldt voor bijna alle complexe toepassingen op het gebied van automatische tekstverwerking.

Dit boek wil althans de instrumenten aanreiken die een reëlere kijk op het gebied mogelijk maken. Instrumenten waarmee men niet meer hoeft te verdrinken in het in- en uitpakken van strings, in het definiëren van bomen of in het bijhouden van pointers. Instrumenten waardoor men niet meer tot de conclusie moet komen dat het onmogelijk is ook maar iets zinnigs met de computer te doen op dit gebied. Een kwestie die naar onze overtuiging en zoals ook uit de resultaten blijkt, vergelijkbaar is met het perspectief van de slak, die weliswaar bij uitstek competent is om de moeilijkheden te beschrijven van zijn slakkegang; die echter incompetent is wanneer het erom gaat de mogelijkheden te beoordelen die verbonden zijn aan de verplaatsing door de lucht van een leeuwerik of zwaluw die een nieuwe lente kan brengen.

HOOFDSTUK 1
ASPECTEN VAN TEXT PROCESSING (TP): OVER VAKBONDEN, EEN ONECHTE VUIST EN GROENE IDEEËN



1.1 "WIJ MAKEN U EROP ATTENT DAT..." of: TP als vijand voor de VAKBEWEGING

TEKSTVERWERKER IN DRIE TALEN VOOR DE DEC SERIE 11

Brussel-Universal Systems, een onderafdeling van het Belgische Sligos Benelux, heeft de alleenvertegenwoordiging van Europa verworven voor het tekstverwerkingspakket Word System-11 van North West Digital Systems. WS-11 is toepasbaar op alle systemen van DEC uit de serie 11 die gebruik maken van RT11, RT11:CTS300, RSTS/E, VAX/VMS en RSX11-M. Het pakket is geschreven in Macro-11 en is in drie talen beschikbaar, te weten Engels, Frans en Nederlands. Naast de gebruikelijke functies van een tekstverwerker omvat Word System-11 het afdrukken en het uitbrengen van lijsten met selecties. Bovendien is het mogelijk rekenkundige bewerkingen uit te voeren en alfabetische en numerieke bestanden in stijgende of dalende volgorde te rangschikken.

Zonder enige twijfel zullen in de komende jaren de meldingen toenemen over 'tekstverwerkers' die steeds meer gaan kunnen en die telkens weer voor een andere machine geschikt zijn. De inleidende tekst van dit hoofdstuk is dan ook niet bedoeld om een extra aanbeveling voor deze ene tekstverwerker, geschikt voor de DEC machine, te doen. Interessant aan deze tekst is te zien dat er kennelijk al 'gebruikelijke functies van een tekstverwerker' zijn. Met tekstverwerker is hier duidelijk niet een menselijk wezen bedoeld, ofschoon in strikte zin alleen de mens teksten kan verwerken.

Toch verraadt de reclametekst voor de DEC-tekstverwerker wel enigszins om wat voor functies het zal gaan. Als men ziet dat er lijsten kunnen worden geproduceerd van selecties met de toevoeging van wat op- en aftelwerk, kan men rustig constateren dat de tekstverwerker standaardbrieven en standaardcontracten zal kunnen schrijven en dat hij verrijkt is met een indexeringsfunctie.

Om zicht te krijgen op wat een tekstverwerker doet, lijkt het een aardig experiment om van de inleidende tekst een standaardtekst te maken. Stel dat de redactie van COMPUTABLE (waar de melding uit genomen werd, Computable, 29 aug. 1980, p. 23) ziet aankomen dat ze iedere week een dergelijke melding moet brengen, dan ziet ze zich voor de keuze geplaatst iedere week opnieuw een typiste grotendeels dezelfde tekst te laten typen, of de tekst in een vorm te brengen waarin alleen het nieuwe, d.w.z. de echte informatie hoeft te worden ingevoegd. Die tekst zou er als volgt uit kunnen zien (met slashes worden de plaatsen aangeduid waar een invoeging plaatsvindt):

Tekstverwerker in / / talen voor / /
/ / / / / / / / /
heeft de alleenvertegenwoordiging van / / verworven voor het
tekstverwerkingspakket / / . / / is toepasbaar
op alle systemen van / / uit de serie / / die gebruik
maken van / / / . Het pakket is geschreven in
/ / en is in / / talen beschikbaar, te weten /

/ / . Naast de gebruikelijke functies van een tekstverwerker omvat / / / / / / / / . Bovendien is het mogelijk / / / en / / / / / / / .

Iedere keer dat een bericht over tekstverwerkers moet worden gedrukt, scheelt dat 50 woorden werk. In die 50 woorden kunnen ook geen schrijffouten meer komen. Verder wordt de aandacht volledig gericht op de echte informatie die in het bericht vervat zou moeten zijn. Men heeft niet erg veel fantasie nodig om te bedenken hoeveel werk in principe kan worden uitgespaard wanneer men werkelijk op zoek gaat naar wat standaardiseerbaar is in teksten van dit soort. Voor de tekst die hier ter sprake was scheelt het ruwweg 40% van het werk (50 woorden standaard, 66 nieuw). Er is naar ons weten geen onderzoek gedaan naar de vraag naar oud en nieuw in teksten volgens onze definitie, maar het ligt zeer voor de hand aan te nemen dat de verhouding in onze voorbeeldtekst nog ongunstig ligt. Er zijn zonder enige twijfel teksten te vinden die nog minder informatie (het nieuwe is 'informatie' in onze gedachtengang) bevatten. Met een klein praktisch voorbeeld willen we dat demonstreren. Iedereen die over de gewoonte beschikt niet alles wat gekocht wordt contant te betalen, moet weleens gemaand worden. Het enige wat voor de leverancier die de aanmaning moet versturen interessant is, is de naam van de schuldenaar, het adres, het bedrag van de rekening en de datum van levering. Met deze gegevens op ponskaart kan een brief automatisch worden verstuurd met de volgende inhoud:

Aan: naam/adres etc.
Wij maken u erop attent dat u uw rekening groot / / sinds / / nog niet heeft voldaan. Wij verzoeken u genoemd bedrag over te maken op onze rekening.
Wanneer u reeds betaald heeft, verzoeken wij u dit bericht als niet verzonden te beschouwen.

Hoogachtend,

Text Processor.

Het computerprogramma dat deze brief schrijft, ziet er in de programmeertaal COBOL als volgt uit:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. BRIEFRAAM.
AUTHOR. BOOT.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. CYBER-175.
OBJECT-COMPUTER. CYBER-175.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT DATA-FILE ASSIGN TO INPUT.
    SELECT PRINT-FILE ASSIGN TO OUTPUT.
```

```
DATA DIVISION.
FILE SECTION.
FD  DATA-FILE
    LABEL RECORDS ARE OMITTED,
    DATA RECORD IS REKENINGEN-REC.
01 REKENINGEN-REC.
    02 NAAM PIC A(14).
    02 ADRES PIC X(24).
    02 PLAATS PIC A(9).
    02 BEDRAG PIC 99V99.
    02 DATUM PIC X(7).
    02 FILLER PIC X(22).
FD  PRINT-FILE
    LABEL RECORDS ARE OMITTED,
    DATA RECORD IS BRIEF.
01  BRIEF.
    02 FILLER PIC X(16).
    02 BRIEFREGEL PIC X(100).
WORKING-STORAGE SECTION.
01 REGEL-1.
    02 FILLER PIC X(16) VALUE SPACES.
    02 FILLER PIC X(43) VALUE "WIJ MAKEN U EROP ATTENT DAT U UW R
-   "EKENING".
01 REGEL-2.
    02 FILLER PIC X(19) VALUE SPACES.
    02 FILLER PIC X(5) VALUE "GROOT".
    02 FILLER PIC X(1) VALUE SPACE.
    02 BEDR PIC $99.99.
01 REGEL-3.
    02 FILLER PIC X(16) VALUE SPACES.
    02 FILLER PIC A(5) VALUE "SINDS".
    02 FILLER PIC X(1) VALUE SPACE.
    02 DAT PIC X(7).
    02 FILLER PIC X(1) VALUE SPACE.
    02 FILLER PIC A(23) VALUE "NOG NIET HEEFT VOLDAAN.".
    01 REGEL-4.
    02 FILLER PIC X(16) VALUE SPACES.
    02 FILLER PIC A(62) VALUE "WIJ VERZOEKEN U GENOEMD BEDRAG OVE
-   "R TE MAKEN OP ONZE REKENING.".
01 REGEL-5.
    02 FILLER PIC X(16) VALUE SPACES.
    02 FILLER PIC A(63) VALUE "WANNEER U REEDS BETAALD HEEFT, VER
-   "ZOEKEN WIJ U DIT BERICHT ALS ".
01 REGEL-6.
    02 FILLER PIC X(16) VALUE SPACES.
    02 FILLER PIC A(29) VALUE "NIET VERZONDEN TE BESCHOUWEN.".
01 REGEL-7.
    02 FILLER PIC X(26) VALUE SPACES.
    02 FILLER PIC A(12) VALUE "HOOGACHTEND,".
01 REGEL-8.
    02 FILLER PIC X(30) VALUE SPACES.
    02 FILLER PIC A(30) VALUE "TEXT PRO CESSOR.".

PROCEDURE DIVISION.
BRIEFRAAM-START.
    OPEN INPUT DATA-FILE OUTPUT PRINT-FILE.
PROG.
    READ DATA-FILE AT END GO TO EIND.
    WRITE BRIEF FROM REGEL-1 AFTER ADVANCING 10 LINES.
    MOVE BEDRAG TO BEDR.
    WRITE BRIEF FROM REGEL-2 AFTER ADVANCING 2 LINES.
    MOVE DATUM TO DAT.
    WRITE BRIEF FROM REGEL-3 AFTER ADVANCING 2 LINES.
    WRITE BRIEF FROM REGEL-4 AFTER ADVANCING 2 LINES.
    WRITE BRIEF FROM REGEL-5 AFTER ADVANCING 2 LINES.
    WRITE BRIEF FROM REGEL-6 AFTER ADVANCING 1 LINES.
    WRITE BRIEF FROM REGEL-7 AFTER ADVANCING 5 LINES.
    WRITE BRIEF FROM REGEL-8 AFTER ADVANCING 3 LINES.
    MOVE NAAM TO BRIEFREGEL.
    WRITE BRIEF AFTER ADVANCING 6 LINES.
    MOVE ADRES TO BRIEFREGEL.
    WRITE BRIEF AFTER ADVANCING 1 LINES.
    MOVE PLAATS TO BRIEFREGEL.
    WRITE BRIEF AFTER ADVANCING 1 LINES.
    GO TO PROG.
EIND.
    CLOSE DATA-FILE PRINT-FILE.
    STOP RUN.
```

Het programma rekent op de volgende input:

```
WILLEMSE, JAN PIET HEINSTRAAT 3          UTRECHT 349011/8/77
PIETERSE, WIM ACHTERBANK 7               HAARLEM 156012/5/77
```

Het programma levert de volgende output op:

```
WIJ MAKEN U EROP ATTENT DAT U UW REKENING

   GROOT $34.90

SINDS 11/8/77 NOG NIET HEEFT VOLDAAN.

WIJ VERZOEKEN U GENOEMD BEDRAG OVER TE MAKEN OP ONZE REKENING.

WANNEER U REEDS BETAALD HEEFT, VERZOEKEN WIJ U DIT BERICHT ALS
NIET VERZONDEN TE BESCHOUWEN.


          HOOGACHTEND,


              TEXT PROCESSOR


WILLEMSE, JAN
PIET HEINSTRAAT 3
 UTRECHT



WIJ MAKEN U EROP ATTENT DAT U UW REKENING

   GROOT $15.60

SINDS 12/5/77 NOG NIET HEEFT VOLDAAN.

WIJ VERZOEKEN U GENOEMD BEDRAG OVER TE MAKEN OP ONZE REKENING.

WANNEER U REEDS BETAALD HEEFT, VERZOEKEN WIJ U DIT BERICHT ALS
NIET VERZONDEN TE BESCHOUWEN.


          HOOGACHTEND,


              TEXT PROCESSOR


PIETERSE, WIM
ACHTERBANK 7
 HAARLEM
```

In het algemeen maatschappelijk verkeer is zeer veel 'lege' tekst te vinden, teksten die slechts bestaan uit vaste formules. Uit 'beleefdheid', uit gemakzucht of uit zelfbehoud. Dit laatste dan vooral wanneer juridische formules in het geding zijn. Zo krijgt men standaardcontracten, -polissen, -statuten, enz. Het is duidelijk dat een consequente toepassing van computers voor de standaardformules in teksten veel werk bespaart of, van de andere kant uit gezien, werkgelegenheid kost, zeker zolang er geen ander werk voor in de plaats gekomen is.

Wat we echter voor ons betoog willen vasthouden is het aspect van TP dat blijkt uit de computertoepassing die boven besproken werd. Voor een gedeelte bestaat tekst uit clichématig gebruikt taalmateriaal. Wanneer men maar de context kent waarin een tekst geproduceerd moet worden, heeft men al snel een serie cliché-uitdrukkingen paraat om het 'standaard-gedeelte' op te vullen. Het clichégedeelte, en dat hebben we willen demonstreren, is vrij gemakkelijk te behandelen door een computerprogramma. En als zodanig hebben we een stukje TP beschreven dat afgezonderd kan worden van wat het betekent, dat 'context-vrij' is, in die zin dat de context om zo te zeggen buiten de deur staat.

Bekijkt men teksten die op een dergelijke manier behandeld kunnen worden wat nauwkeuriger, dan komt men al ras tot de ontdekking dat ze zeer vervelend zijn. Dat komt door de voorspelbaarheid van wat er komen gaat. Daardoor is de tekst als het ware fysiek voorspelbaar. Van dit kenmerk maakt de normale tekstverwerker gebruik. Voor ons betoog hebben we nu geïsoleerd de factor: de FYSIEKE STRUCTUUR van een tekst. De fysieke structuur van een tekst is gekoppeld aan het aspect 'voorkomen', een 'statistisch' aspect. Hoe meer de aandacht op het fysieke aspect van een tekst komt te liggen, hoe minder inhoud hij krijgt, hoe vervelender hij wordt en hoe sneller er een tekstverwerkend programma voor gevonden wordt. Voor het fysieke aspect van tekstverwerking zijn dan ook vele standaardprocedures geschreven. We zullen er in hoofdstuk 2 op terugkomen.

1.2 "BLUT IST GANZ EIN BESONDERER SAFT" (Goete, 'Faust', 1740) ofwel: niet iedere Faust is een vuist

Werd in het vorige hoofdstukje een 'lege' tekst gedemonstreerd, de nu volgende tekst is het tegendeel van leeg:

MEFISTOFELES:
Vergeef mij, ik heb geen verheven woorden,
Al zie ik ook de spot om aller mond;
En stellig zoudt ge, als gij mijn pathos hoorde,
Zelf moeten lachen, zo gij lachen kondt.
Ik weet van zon noch sterren te gewagen,
Ik zie de mensen slechts en al hun plagen.

De kleine god op aarde blijft van 't zelfde slag,
Hij 's even wonderlijk als op de eerste dag.
Hij zou nog wel iets beter leven,
Hadt gij hem niet een vonk van 't hemellicht gegeven.
Hij noemt het rede en dat is
Voor d'ergste dierlijkheid een dun vernis.
Hij komt mij voor, vergeef' het Uw Genade,
Als een van die langbenige cicaden,
Die telkens wel wat vliegt en springt,
Maar steeds in 't gras het oude liedje zingt.
En zat hij dan nog maar in 't gras gedoken,
Doch in de mest heeft hij zijn neus gestoken.

DE HEER:
Hebt gij mij verder niets te zeggen?
Moet g'altijd iets ten laste leggen?
Staat er dan nooit op aarde u iets aan?

MEFISTOFELES:
Neen, Heer, ik vind er nog altijd niets gedaan.
'k Ben met de mens begaan in zijn ellende,
Mis zelfs de moed hem nog meer leed te zenden.

HEER:
Kent gij Faust?

MEFISTOFELES:
Doctor Faust?

DE HEER:
Mijn onderdaan!

MEFISTOFELES:
Voorwaar, hij dient u op bijzondre wijze,
De dwaas is wars van aardse drank en spijzen.
Door diepe onrust voortgedreven,
Is hij zijn dwaasheid zich maar half bewust;
De helste ster moet hem de hemel geven,
Op aarde jaagt hij naar de hoogste lust,
Hij heeft zich her en der begeven,
Maar nergens vond zijn zoekend hart nog rust.

DE HEER:
Al vindt hij thans de weg naar mij nog niet,
Haast wordt zijn dwaling door mijn licht bezworen.
De tuinman, die het boompje groenen ziet,
Vertrouwt dat bloem en vrucht hem zijn beschoren.

MEFISTOFELES:
Waarom gewed? Hij gaat voor u verloren!
Wanneer gij mij vergunning geeft,
Zal ik hem gaandeweg bekoren.

DE HEER:
Zolang hij op de aarde leeft,
Is zulk een poging niet verboden.
Dwaalt niet de mens, zolang hij streeft?

MEFISTOFELES:
Mijn dank daarvoor, want met de doden
Wist ik nog nooit wat aan te vangen.
Ik houd toch maar het meest van frisse, volle wangen.
Als 't om een dode gaat, ben ik niet thuis.
'k Houd als de kat van 't spelen met de muis.

DE HEER:
Nu goed, gij hebt verlof ontvangen!
Ga nu en voer die geest dus uit zijn baan,
En laat hem, als gij hem kunt vangen,
De weg met u bergafwaarts gaan.
Erken dan, als uw rekening niet deugt,
Beschaamd: hoe duister het ook moge wezen,
Een goed mens vindt het pad terug der deugd.

MEFISTOFELES:
Ja, als 't maar duurzaam wilde wezen!
Ik hoef niet voor mijn weddenschap te vrezen.
En wordt de palm mij toegewezen,
Gunt van de zege dan de volle vreugd.
Stof zal hij vreten, dat 't hem heugt,
Gelijk mijn moei, de slang zo hoog geprezen.

DE HEER:
Hier kunt g' u ongemoeid vertonen.
U en de uwen heb ik nooit gehaat.
Het is in 't heir van de demonen
De spotgeest die mij 't minste tegenstaat.
De mens is gauw geneigd zijn plichten te verzaken,
Hij voelt zich licht in ledigheid tevree;
Daarom geef ik hem graag die makker mee,
Die 't scheppingswerk als duivel helpt volmaken.
Maar gij, o ware godenzonen,
Verheugt u over al het levend schone!
De schepping, die door eeuwig worden leeft,
Zij moge u in de sfeer der liefde hullen,

Tracht gij, wat andren vaag voor ogen zweeft,
In blijvende gedachten te vervullen!

MEFISTOFELES (alleen):
Ik zie de Oude graag van tijd tot tijd,
En wacht mij wel met hem te breken.
Hoe vriendlijk van hem uit zijn heerlijkheid
Zo menslijk met de duivel zelf te spreken.

Voorspelbaar aan deze tekst is tot op zekere hoogte slechts één enkel puntje, namelijk de klank van het laatste woord van iedere regel. Verder tot op zekere hoogte maar niet compleet, de lengte (aantal lettergrepen) van iedere regel. Toch is de tekst verre van onbegrijpelijk, tenminste wanneer men iets weet van het spel tussen 'goed' en 'kwaad' en van sommige opvattingen omtrent de rol van de mens als bemiddelaar tussen 'hemel' en 'hel'. Men heeft plezier om de gevatte antwoorden van Mefisto. Gevat, omdat ze plotselinge wendingen geven. De wendingen zijn puur rationeel van aard. Verder is Mefisto een meester in spelletjes met woorden, of liever met inhouden, want men kan hem niet bepaald een woordfetichist noemen. Nog veel meer zou men over deze tekst kunnen opmerken. Al die opmerkingen zouden opmerkingen zijn over inhoudelijke zaken die in de tekst aanwezig zijn, uit de tekst kunnen worden geconcludeerd (bijvoorbeeld de wereldbeschouwing van de auteur) of die men zelf in de tekst legt (de een vindt een bepaald grapje leuk, de ander niet, enz.). Men zou, wat in literaire studie aan universiteiten wel voorkomt, aan deze tekst een half jaar studie kunnen wijden. Daarbij zou het praktisch helemaal gaan over zaken die met de fysieke kant van de tekst slechts zeer weinig direct van doen hebben. Mocht men de fysieke kant willen benaderen dan houdt het wel zo ongeveer op bij het uitzetten van de klinkerverdelingen over de pagina, of bij het vervaardigen van een lijstje met in de tekst voorkomende woordvormen. Fysiek is deze tekst volgens geen enkele standaard dus leeg. Inhoudelijk is de tekst vol. Zo vol dat het een genoegen is je er keer op keer mee bezig te houden.

Voor ons betoog hebben we nu geïsoleerd de factor INHOUD. We hebben daarvoor een tekst genomen waarbij de fysieke structuur nauwelijks relevant was. Het verschil met de 'standaardtekst' waarin de fysieke structuur een belangrijke rol speelt, is dat de informatie in de tekst niet op eenvoudige wijze gelokaliseerd kan worden. Het is niet voorspelbaar waar wat moet worden ingevoegd op vergelijkbare wijze als met de standaardbrief uit ons voorbeeld het geval was. Bij het isoleren van wat we tot nu toe inhoud hebben genoemd is wel gebleken dat inhoud voor de nu genoemde tekst verre van kernachtig geformuleerd, laat staan gedefinieerd kan worden. Met deze tekst in de hand kan men zich vermijen over de precieze uitspraken die gedaan worden, men kan plezier hebben over de gevatheid van de antwoorden van Mefisto, maar men kan evenzogoed diepzinnige beschouwingen gaan houden over het

'wereldbeeld' dat spreekt uit dit fragment van Goethe's Faust, of over de verhouding van goed en kwaad in de schepping. Het aspect 'inhoud' dat we nu hebben geïsoleerd heeft zodoende implicaties die ver buiten de tekst zelf reiken. De tekst heeft om zo te zeggen een verwijzende functie naar allerlei gebieden van kennis: esthetische, maar ook filosofisch/religieuze om er maar een paar te noemen. Verder is de neiging gebleken allerlei gevolgtrekkingen (interferenties) uit de tekst te maken die 'inhouden' hebben, die niet expliciet in de tekst staan aangeduid. Stellen we tekst voor als een lijn met een bepaalde lengte, dan moeten de twee begrippen die we nu gevonden hebben, te weten fysieke structuur en inhoud, zover mogelijk van elkaar op die lijn worden aangebracht; een uitgebreider verschil dan tussen deze twee kenmerken van tekst is niet voorstelbaar. Wel is het beeld van de lijn goed om uit te drukken dat er een relatie bestaat tussen de tekst als fysieke eenheid (bijvoorbeeld van woorden) en de tekst als beeld van 'inhouden'. Die relatie is alledaags van aard: is er fysiek geen tekst dan kan men er ook geen inhouden mee verbinden. Men kan dus een verband vermoeden dat verloopt vanuit de fysieke tekst naar de inhoud. Als volgt:

FYSIEKE STRUCTUUR ⟶ INHOUD

Dit zou men kunnen zien als een soort causale keten die één kant uit gericht is. Deze relatie geldt echter alleen wanneer men uitsluitend oppervlakkig redeneert: zonder tekst als serie woorden vastgelegd is er geen inhoud. Deze redenering is oppervlakkig omdat tekst niet uitsluitend kan worden vastgelegd als een serie woorden. We komen hier later op terug. De redenering is ook oppervlakkig omdat het verband niet uitsluitend van links naar rechts verloopt bij alle teksten. Er zijn ook teksten waarin de inhoud of afzonderlijke inhoudelijke aspecten sterke implicaties hebben voor de fysieke structuur. In de voorbeeldtekst uit Goethe's Faust zou men met enige spitsvondigheid een 'zwak' inhoudelijke functie kunnen afleiden uit fysieke kenmerken als rijm en lengte van de regels of de versmaat die gebruikt werd. Een duidelijker voorbeeld echter van het belang voor de inhoud dat kan worden gevonden in de fysieke structuur van een tekst geeft bijvoorbeeld de volgende tekst:

Want weet je

Want weet je wat de liefde is
De parel in een rotte vis

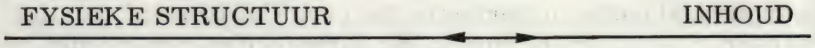
Al zou ik dus met hondetongen spreken
En deed ik niet aan vis

Bij God ik zou niet weten
Wat wel
De liefde is

De fysieke structuur van deze tekst is of lijkt althans heel eenvoudig. Een reeks zinnen bestaande uit hoofd- en bijzin, waarvan ieder apart deel (zin) een eigen regel krijgt. Het geheel wordt omsloten door een niet complete zin (want weet je ... wat wel de liefde is). Doordat echter de invulling van de eerste onaffe kreet (want weet je = op zich ook weer een stukje uit de spreektaal dat dikwijls apart staat en niet verder wordt afgemaakt) zolang wordt uitgesteld valt de naïeve lezer dit gedeelte van de structuur doorgaans niet op. Daardoor valt ook niet direct op dat een gedeelte van de eerste regel en de laatste regel identiek zijn op één woord na. Dat ene woord krijgt (als men het belang ervan uit de fysieke structuur heeft afgelezen) bijzondere nadruk. Die nadruk wordt eveneens weerspiegeld in de manier waarop het woord ('wel') in het gedicht terechtkomt. Het staat als enige binnen het gedicht aan het eind van de regel zonder dat de zin en/of een zinsdeel uit is. Daardoor ontstaat achter het woord wel een pauze. De nadruk op het woord 'wel' en de pauze achter het woord kunnen het zijn afgesleten functie als bijwoord ontnemen en sommige lezers eraan herinneren dat 'wel' ook voorkomt in de uitdrukking 'wel en wee' of liever nog dat 'wel' ook een zelfstandig naamwoord is dat betekent: plaats waar water uit de grond opborrelt.

Vanaf die ontdekking is de ruimte weer volstrekt open voor allerlei andere invullingen zoals dat ook kon bij de tekst uit Goethe's Faust.

Voor ons betoog is echter van belang dat gedemonstreerd of althans enigszins aannemelijk gemaakt kon worden dat de fysieke structuur van een tekst, vastgelegd als een serie uiterlijke en tastbare kenmerken als verdeling van de woorden over het papier, diepe gevolgen kan hebben voor de inhoud van de tekst en dat de relatie fysieke structuur/inhoud verre van triviaal hoeft te zijn. Of men die implicaties overigens zelf ook wil oppakken, of men de interpretatie van 'wel' in dit gedicht al of niet meeneemt, is een heel andere zaak. Deze kwestie doet hier niet ter zake. We besluiten daarom met te stellen dat de relatie tussen fysieke structuur en inhoud naar twee kanten wijst. Grafisch als volgt:



De fijnheden en spitsvondigheden of fijnzinnigheden die nodig zijn om de inhoud volledig te vinden in een tekst zijn vandaag de dag niet in computerprogramma's in te planten (implementeerbaar). We zullen in dit boek dan ook, hoe spijtig dat wellicht ook is, niet veel aandacht meer aan de extreme rechterkant van de afbeelding besteden.

1.3. "COLOURLESS GREEN IDEAS SLEEP FURIOUSLY" (Chomsky, Syntactic Structures, 1965, p. 15)

In het voorafgaande werd afgestevend op indeling van fysieke structuur en inhoud in een tekst. Daarbij moest worden geconstateerd dat de inhoud weliswaar minder grijpbaar is dan de fysieke structuur van een tekst, maar dat er toch tweerichtingsverkeer tussen die twee componenten bestaat. In dit hoofdstukje wordt het begrip 'fysieke structuur' aan een nader onderzoek onderworpen.

Fysieke structuur werd tot nu toe bedoeld als datgene in de tekst waaruit kon worden afgeleid waar nieuwe en waar standaard- of oude informatie uit afleesbaar was. Daarbij hebben we 'informatie' gehanteerd in de betekenis van 'mededeling'. Dit is het niveau waarop de niet taalkundig geschoolde met een tekst omgaat. Het is echter niet toevallig dat we steeds hebben gesproken van fysieke s t r u c t u u r. Men kan moeilijk een term als structuur hanteren wanneer het werkelijk alleen nog maar gaat om invuloefeningen van oude en nieuwe gegevens. Ook op het niveau dat onder de mededeling zelf ligt bestaat het kenmerk: oude informatie/nieuwe informatie. De woorden die bijvoorbeeld in een zin voorkomen zijn niet allemaal even sterk nodig voor het doorgeven van boodschappen. Men kan bijvoorbeeld rustig zeggen:

ikke niet verstaan

in plaats van

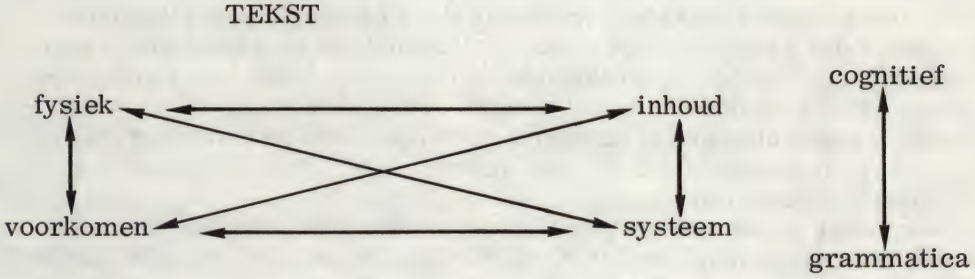
ik begrijp niet wat je bedoelt.

De woordenschat (het lexicon) van een taal bevat een serie woorden die de structuur van de zin aangeven. Dat zijn woorden als: 'de', 'het', 'is', 'je'. Ze worden heel toepasselijk structuurwoorden (ook wel grammaticale woorden) genoemd. Deze woorden hebben nauwelijks een reële, op de werkelijkheid betrokken betekenis. Ze schrijven voor hoe een zin in elkaar gaat zitten waarin de mededeling over de werkelijkheid gedaan gaat worden. Men kan deze eigenschap echter ook omkeren: wanneer men weet waar de structuurwoorden in een zin staan, dan weet men ook de plaatsen te vinden waar de 'inhoudswoorden' van die zin staan. De structuurwoorden hebben duidelijk dezelfde functie als de stukken standaardinformatie die boven werden gebruikt om een standaardbrief te programmeren. Ze verwijzen nu echter niet naar iets echt oppervlakkigs, maar naar de grammatica van een taal, met name naar de zinsbouw. En daarmee zouden we nu al aangekomen zijn bij hoofdstuk 3 van dit boek waarin wordt behandeld hoe men die inhoudswoorden en de structuurwoorden door de computer laat vinden/aanmaken en waarin ook behandeld wordt hoe men computerprogramma's voor zinsontleding kan schrijven. Het zal duidelijk zijn dat dit een belangrijke verworvenheid van een textprocessor moet zijn

voor hij werkelijk text-processor genoemd kan worden. Jammer is alleen dat de mens zelf deze taken zo volstrekt routinematig en onbewust oplost, dat hij aanneemt dat het ook taken zijn die verwaarloosbaar zijn voor computers. Reden waarom de ideeën achter de gemiddelde tekstverwerkingspakketten eerder kleurloos dan groen zijn.

Samenvattend kan het volgende worden gesteld over tekst.

Iedere tekst heeft een fysiek aspect. Dit aspect behelst het voorkomen van afzonderlijke eenheden in de tekst. Eenheden heeft men op verschillend niveau: een tekst bevat letters, woorden, zinsdelen, zinnen, alinea's, hoofdstukken, enz. Deze eenheden hebben niet uitsluitend een volgorde-verband met elkaar, ze hebben ook een structureel verband. Dit laatste wordt voor de zinsbouw gedefinieerd in de grammatica. Voor bijvoorbeeld een redenering vindt men de syntactische (d.w.z. structurele) definitie in de logica. Voor bijvoorbeeld een sonnet vindt men het structurele verband geformuleerd in de versleer. Kortom: het fysieke aspect van de tekst is verbonden met een systematisch aspect. Datzelfde geldt voor het inhoudelijke aspect, dat in laatste instantie verbonden is met denksystemen (cognitieve), maar ook met culturele (denk-) systemen. Daarom zouden we de grafische voorstelling van de lijn tussen fysieke structuren en inhoud als volgt willen uitbreiden:



1.4. COMPUTERS EN COMPUTERTALEN VOOR TEXT PROCESSING

Omgaan met teksten wil zeggen omgaan met letters, woorden, zinnen, hoofdstukken, inhouden, denksystemen. Aan teksten valt iets te tellen, er vallen overzichten te maken, bijvoorbeeld van welke woorden een auteur gebruikt. Maar ook moet men structuren van zinnen en inhoudelijke zaken in teksten kunnen terugvinden. Text Processing omvat dus zowel echt rekenen en boekhouden als nabootsing van intelligent gedrag. Het spreekt voor zich dat er een bijna onoverbrugbare kloof ligt tussen '1 en 1 is twee' en het onderwerp van deze zin is 'het'. Computers zijn machines die erg eenzijdig zijn. Voor computertalen is het werkelijk

onoverbrugbaar om van een som naar een ontleding te komen. Dit vindt zijn weerslag in het ontstaan van een onoverzichtelijke menigte van computertalen. Op zich werden al die talen ontwikkeld voor aparte toepassingsgebieden. De twee standaardgebieden zijn: rekenen en administratie bedrijven. Of wetenschappelijk werk en commercieel werk. Deze tweedeling is niet toevallig ontstaan maar uit de nood van de computer zelf geboren. In de loop van de geschiedenis van de computer heeft men op een gegeven ogenblik deze natuurlijke groei gehouden voor een miskraam. Men bedacht dat men computertalen en computers moest gaan ontwikkelen die alles even gemakkelijk aankonden. Zo ontstond uit een samenvoeging van de structuur van de op rekenwerk ingestelde taal ALGOL60 (van ALGOrithmic Language 1960) en de op administratieve toepassingen toegesneden taal COBOL (van COmmon Business Oriented Language) de 'language for all seasons': PL/I. Inderdaad is dit een handige taal die snel en efficiënt werk mogelijk maakt dat in de aparte talen ALGOL60 en COBOL zeer omslachtig te organiseren is. Maar hoe meer men zijn aandacht richt op inhoudelijke zaken, hoe moeilijker ook een taal als PL/I zich gaat gedragen. PL/I blijft de programmeur belasten met veel kennis over de ingewanden van de computer, zodat de programma's toch weer zo lang moeten worden dat de animo of het geld op is voor men aan zijn werkelijke probleemstelling toekomt. Met PL/I heeft men dan nog het gunstigste voorbeeld uit de oude programmeertraditie van computers in de jaren 1965-1970. Alle programma's in andere talen van deze categorie zijn langer en daardoor ook geheimzinniger voor de buitenstaander. Hem wordt niet duidelijk dat de lengte correspondeert met de domheid van de computer, hij denkt eerder dat het zijn eigen domheid is wanneer hij wil protesteren tegen die rare programma's waarin je letters moet in- en uitpakken. Hoe meer de computer geleerd heeft, hoe korter de programma's kunnen zijn.

De winst uit de vroege jaren van computertoepassingen met talen als FORTRAN en PL/I is dat er nu voor het maken van woordenlijsten en tellingen van woorden een serie standaardprogramma's aanwezig is die men zo kan gebruiken. Gebruik van standaardprogramma's veroorzaakt nu inderdaad het korte programma dat niet meer belast is met het krullejongenswerk, waar zoveel inspanningen tot nu toe zijn ingekropen. Merkwaardig is wel dat deze standaardprogramma's niet algemeen gebruikt worden en dat men toch steeds bezig gaat om weer opnieuw een programmaatje te schrijven in de oude talen. Ja, men verzint er zelfs helemaal als mosterd na de maaltijd nog nieuwe Oude talen bij die universeel zouden zijn. Een treurig voorbeeld daarvan is de taal ALGOL68, de opvolger van ALGOL60, waarin de meeste concepten van PL/I nog eens, en nu 'netjes' werden ingebouwd. Daardoor werd ALGOL68 een taal waarin men op een ingewikkelde manier nog eens het wiel kan gaan uitvinden; een taal ook die flink duur in gebruik is en die de gebruiker dwingt niet over de problemen te denken die hij wenst op te lossen, maar of hij wel programmeert in de lijn van bepaalde concepten uit de theoretische informatica.

Uit het betoog tot nu toe mag duidelijk zijn dat wij niet veel zien in de aanprijzing van één programmeertaal die voor alles goed zou zijn. Wil men textprocessing aan de basis bedrijven en niet alleen gebruik maken van standaardpakketten, dan zal men zich in meer dan een taal moeten bekwamen. Men zal een 'rekentaal' moeten uitzoeken en men zal een programmeertaal moeten uitzoeken waarmee men ook niet-numeriek werk kan verrichten. Ook zal men zich vlug moeten leren oriënteren in het gebruik van standaardprogrammatuur op de genoemde gebieden. Aangezien er bij het werken met teksten nogal eens de wens bijkomt dat er op handige wijze in- en uitvoer van de te bewerken of de bewerkte gegevens mogelijk moet zijn, is de meeste programmatuur van het 'rekensoort' in dit boek geschreven in PL/I. Ook een enkele toepassing op inhoudelijk niveau is geprogrammeerd in PL/I. De meeste toepassingen op inhoudelijk niveau zijn echter ontwikkeld op het terrein van de kunstmatige intelligentie. Daar gebruikt men de programmeertaal LISP. Dat is dan ook de reden dat aan LISP in dit boek de meeste aandacht wordt besteed. Deze taal heeft niet de nadelen die boven ten aanzien van de 'ALGOL-achtige talen' werden opgenoemd. Men hoeft er geen letters of woorden in uit en in te pakken. Men hoeft geen pointers bij te houden om te weten waar men ook al weer wat in het computergeheugen had neergezet en men hoeft niet zelf een administratie te gaan verzinnen voor het werken met boomstructuren. Daarnaast zal blijken dat men voor veel toepassingen op het gebied van TP een patroonherkennend programma nodig heeft. Als een dergelijk programma kan de in LISP geschreven programmeertaal METEOR worden beschouwd. Ook aan deze taal zal ampel aandacht worden besteed.

Voor de volledigheid moet nog worden gewezen op de programmeertaal SNOBOL. Mocht men niet over LISP beschikken dan zou deze taal een goed alternatief zijn. Ook hier heeft men veel minder dan in de ALGOL-achtige talen te maken met de ingewanden van de computer wanneer men met teksten wil werken. Het is echter niet waarschijnlijk dat men wel over SNOBOL en niet over LISP beschikt, reden waarom aan SNOBOL maar nauwelijks aandacht wordt besteed in dit boek (zie hoofdstuk 3).

TESTVRAGEN EN OPGAVEN

1. Uit hoeveel functionele delen bestaat het COBOL-programma voor de vervaardiging van een standaardbrief (zie p. 10).
2. Met welke regels wordt de in- en uitvoer geregeld?
3. Waar bevindt zich het standaardgedeelte van de brief in het programma?

4. Hoe brengt men spaties aan in een regel?
5. Hoe kan men aangeven dat een tekstregel uit het programma doorloopt? (Dit doorlopen moet geschieden wanneer de tekst kolom 72 op de ponskaart heeft bereikt!)
6. Hoe geeft men in COBOL aan dat een rekenopdracht voor de computer beëindigd is?
7. Hoe komt het programma te weten wat er moet gebeuren als er geen invoergegevens meer zijn?
8. Hoe verzorgt men de interlinie (= regels overslaan) in de te printen tekst?
9. Schrijf een programma voor een standaardbrief met de volgende inhoud:

> Geachte ...,
>
> Wij zijn niet gewend om op contactadvertenties te reageren, maar omdat het vandaag /datum invoegen/ is en bovendien /dag van de week invoegen/ hebben wij van dit principe afgezien.
> Ook hebben wij een bedrag groot /$ / in de lotto gewonnen. We nemen dus aan dat u met graagte langs deze onsympathieke weg naar ons terugschrijft om met ons in contact te komen. /toepasselijk spreekwoord invoegen/
>
> Hoogachtend,
>
> Guus en Marie Geluk.

LITERATUUR

Berkeley, E.C. & D.G. Bobrow (ed.):
"The programming language LISP: its operation and applications." M.I.T. Press, 1964.

Boot, M.: "Die Algorithmisierung linguistisch/literarischer Fragestellungen." Utrecht, 1975.

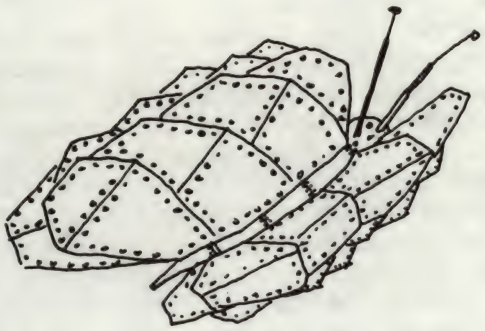
Boot, M.: "Comparable computer languages for linguistic and literary data processing: performance." In: ALLC Bulletin, 1978, vol. 6, no. 2, en 1979.

Boot, M.: "Comal: Computers voor Alfaas, een algemene inleiding." Utrecht, 1975.

Chomsky, N.: "Syntactic Structures." Amsterdam, 1965.

Dijk, T.A. van: "Tekstwetenschap, een interdisciplinaire inleiding." Aula, 633.
Goethe, J.W. von: "Faust, I." Jubil. Ausgabe, Cotta, 1940.
Griswold, R., J.F. Poage & I.P. Polonski: "SNOBOL 4."
Higman, B.: "A comparative study of programming languages." Elsevier, 1967.
Koppelaar, H.: "De programmeertaal van de toekomst." Stam, 1980.
Poel, W.A. van der & L. Maarssen (eds.): "Machine oriented higher level languages." North Holland, 1974.
Pollack, S.V. & T.D. Sterling: "A Guide to PL/I." 1969.
Sammet, J.E.: "Programming languages: history and fundamentals." Prentice Hall, 1969.

HOOFDSTUK 2
STAAR, FIETSEN, HOW IS YOUR SEX LIFE: OVERZICHT VAN REEDS ONTWIKKELDE PROGRAMMATUUR VOOR TEXT PROCESSING



Er is geen gebied van text processing te vinden of er zijn wel computerprogramma's beschikbaar waarmee een gedeelte van de problematiek of zelfs het hele probleem wordt opgelost. Hoe dichter het probleem kan worden gelegd tegen en gedefinieerd aan de fysieke structuur van de tekst, hoe verder de oplossing van het probleem gevorderd is. Daarnaast echter worden op teksten operaties verricht die samenhangen met het feit dat tekst altijd omvat: VEEL. In een boek over text processing moet dan ook een korte indruk worden gegeven over computeroperaties die op VEEL worden toegepast, n.l. de procedures die gebruikt worden in de statistiek. Er is een gerenommeerd pakket van standaardprocedures voor statistisch gebruik in de sociale wetenschappen, genaamd Statistical Package for the Social Sciences (SPSS). Dit hoofdstuk bevat een korte kennismaking met SPSS. Daarnaast bevat het hoofdstuk een overzicht van ontwikkelde programmatuur voor 'inhoudelijk' onderzoek van teksten. De stand van zaken is op dit gebied nog niet zo dat er echt veel beroemde pakketten zijn zoals SPSS. Wel zijn er natuurlijk de commerciële pakketten die in hoofdstuk 1 werden behandeld.

Alleen voor het maken van woordenlijsten uit teksten bestaat een serie programma's die vergelijkbaar zijn met SPSS. Ze hebben ook de mogelijkheid output op te leveren die direct door SPSS kan worden behandeld. De andere gebieden, te denken valt aan automatische inhoudsanalyse, aan automatisch de krant lezen, aan automatisch vertalen, aan automatisch woorden uit teksten halen, aan automatische zinsanalyse en tekstproduktie en wat men niet nog allemaal bedenken kan, deze andere gebieden leveren het beeld op van een weelderig bloeiend veld met een uiterst onregelmatig oppervlak. Dit hoofdstuk geeft in het kort aan hoe men een pad door het veld kan vinden. Doelstelling daarbij is vooral dat men er niet toe wordt verleid zelf opnieuw het tekstwiel te gaan uitvinden, en bijvoorbeeld weer opnieuw zelf een programma gaat schrijven dat woorden in een tekst opzoekt.

Bij de behandeling van de verschillende programmatuur wordt slechts summier ingegaan op de theoretische achtergronden, de aandacht valt op de programmatuur zelf. Dat is de reden dat in de appendix bij dit hoofdstuk verschillende uitgeteste computerprogramma's worden afgedrukt.

2.1 STAAR, STATISTIEKEN EN TEKSTEN

Tellen aan teksten gebeurt meestal op een enkel niveau van wat er aan teksten waar te nemen valt. Zo heeft de beroemde Kaeding voor het Duits de eerste frequentietelling van woorden gemaakt om er achter te kunnen komen, hoe het zat met de letterverdeling van het 'gemiddelde' woord in het Duits. Daaruit kon dan weer een efficiënte manier van

kortschrift (stenografie) worden gedestilleerd voor het Duits. Lettergrepen zijn nog nauwelijks geteld omdat het tot nu toe praktisch onmogelijk was een computerprogramma te schrijven dat foutloos woorden afbreekt (zie echter hoofdstuk 3). De klanken van een taal, de klanken die voorkomen in een tekst zijn evenmin op deze manier telbaar tot nu toe (zie echter hoofdstuk 3). Wilde men in het verleden rijmwoorden tellen, wat van belang is bij bepaalde literaire genres, dan codeerde men de woorden met de hand. Woorden zijn ook en zelfs veelvuldig geteld, te denken valt aan de frequentietellingen van het Amerikaans-Engels van Kucera en Francis, genoemd het Brown Corpus, naar de universiteit die dit project financierde en liet uitvoeren. Ook van het Brits-Engels (het Lancaster Corpus/het Lund Corpus), van het Duits (Meier), maar ook weer Kaeding, van het Nederlands (Uit den Boogaart), eigenlijk van alle talen uit rijke landen zijn woordtellingen gemaakt. Deze tellingen zijn verricht op de ruwe woordvorm, dat wil zeggen mocht een woord in die gedaante meer dan één betekenis hebben dan vindt men dat niet terug in de lijsten. In de frequentietelling van het Nederlands is men zo rigide bezig geweest dat zelfs woorden die met een hoofdletter worden geschreven, dat wil zeggen voor de computer een andere vorm hebben, worden onderscheiden van dezelfde woorden zonder hoofdletter. Men vindt dus 'Het' en 'het' in de frequentietellingen van het Nederlands als verschillende woorden. Kritiek is er genoeg te geven op de grote frequentietellingen die tot nu toe zijn uitgevoerd. Inhoudelijke, maar ook technische. De programmatuur voor het maken van frequentietellingen is meestal ingewikkeld doordat er computertalen voor worden gebruikt die niet geschikt zijn voor stringmanipulaties, een taal als FORTRAN of ALGOL bijvoorbeeld.

Toch is het belangrijk statistische operaties op teksten te kunnen toepassen: teksten hebben immers een 'uiterlijk' aspect. Ook is het zo dat bijvoorbeeld woorden of klanken die veel voorkomen gemakkelijker worden verstaan of begrepen dan woorden en klanken die niet veel voorkomen. Voor allerlei vraagstellingen die samenhangen met taalverwerking en taalleren is het van belang een inzicht te hebben in de statistische structuur van teksten. Zoals gezegd vinden studies in dit opzicht meestal plaats op woorden. Men staart zich blind op woorden, wat noodzakelijk is zolang de problemen die text processing hier oplevert nog niet zijn opgelost. Om tellingen op woorden te kunnen doen moeten ze eerst worden gelokaliseerd in de lopende tekst. Een operatie die met een eenvoudige standaardprocedure kan worden gedaan. Het vinden van woorden in een tekst wordt doorgaans aangeduid met s c a n n e n . Het programma dat de woorden vindt heet SCANNER .

SCANNER

We bieden hier een aparte scanner aan, omdat een SCANNER kan worden gebruikt om alle informatie die in schrift ligt opgeslagen vast te leggen, zo kan men meenemen of een woord met een hoofdletter werd geschreven en zodoende een specifieke syntactische functie heeft (bijvoorbeeld eigennaam of begin van een nieuwe zin). Er zijn computertalen waarin om de losse woorden in een tekst te vinden geen aparte SCANNER nodig is. Een dergelijke taal is LISP, die in extenso wordt behandeld in dit boek. Wil men echter een aparte s c a n - fase om daarmee geheel andere operaties op losse woorden te beginnen dan is het handig een aparte SCANNER te hebben in een taal waarin men op eenvoudige wijze in- en output kan plegen. Een dergelijke taal is PL/I, die bovendien zeer wijd verbreid is.

- Hoe de PL/I tekst globaal te lezen?

Het programma BOOTEXT bevat een serie handelingen die logisch bij elkaar horen. Het programma moet teksten kunnen 'inlezen'. Dit gebeurt met de opdracht:

GET ... EDIT (zie opdracht nr. 84)

Ook moet het programma de gemaakte woorden weer kunnen 'wegschrijven'. Dat gebeurt met de opdracht:

PUT ... EDIT (zie opdracht nr. 39)

Daarnaast moet het programma bepalen waar wat in de 'input stroom' staat. De input stroom is gedefinieerd als een reeks van eenheden die bestaan uit 80 tekens (d. w. z. gewone ponskaarten, genoemd 'het kaartbeeld'). Deze staan op een stuk geheugen dat heet

DAFDU (zie opdracht nr. 67)
OPEN FILE (DAFDU) STREAM INPUT LINESIZE (80)

Weggeschreven wordt op het stuk geheugen dat heet

CELAN (zie de opdracht in regel 67 voor de definitie van CELAN en voor een wegschrijfopdracht zie opdracht nr. 38)

Daarnaast bevat het programma een serie PROCEDURES. Deze procedures bevatten handelingen die logisch bij elkaar horen. Zo moet de computer intern vastleggen van waar tot waar een speciaal woord op

de kaart loopt. Dat gebeurt in de PROCEDURE ALLA (5-28). Ook is er een procedure die de tellers (die bijhouden waar de inlezer van de computer gebleven was op de kaart en die de lengte van het woord bijhouden) weer op een beginwaarde zet voor een nieuw woord. Dit gebeurt in de PROCEDURE OLLA (29-33). Verder moet het programma natuurlijk iedere letter controleren en uitmaken of die letter een leesteken is of een gewone letter (117-178). Aan het eind van het programma wordt opgegeven hoe de inhoud van de informatie op CELAN opgebouwd is (zie de opdrachten 193-195).

Deze aanduidingen zijn voldoende om een globale indruk te krijgen van de opbouw en de werking van de SCANNER. Een karwei als woorden lezen is voor de computer kennelijk niet zo eenvoudig als het voor mensen is die kunnen lezen. In feite staat men met de computer iedere keer weer voor het punt dat hem alles opnieuw geleerd moet worden, tenzij men een zeer hoge programmeertaal als bijvoorbeeld LISP gebruikt. Dat is de reden dat men telkens opnieuw voor het probleem komt te staan dat men zelf het wiel weer gaat uitvinden. Dat uitvinden zelf is best wel een karwei dat inventiviteit en tijd vergt, zoals duidelijk geworden zal zijn uit de kennismaking met de SCANNER die boven werd besproken. Het is tevens de reden dat er weinig werkelijke aandacht overblijft voor de problemen die na de uitvinding overblijven. Toch zijn dat de werkelijke problemen die horen bij t e k s t e n. We nemen van nu af aan maar aan dat de tekstmachine die we maken gebruik maakt van een SCANNER als BOOTEXT en dat de output van die scanner er uitziet als weergegeven op p. 32.

De informatie die de SCANNER oplevert kan men globaal samenvatten met de term: grafematische informatie. Alle informatie die opgeslagen is in het fysieke beeld van het s c h r i f t wordt ontdekt en in een handzamer (voor de computer) vorm onthouden. Men kan uit de output file afleiden welke woorden, maar ook welke zinnen, wat voor soort zinnen, wat voor soort subzinnen, maar ook welke plaats de zin in het geheel en het woord in de zin of subzin inneemt. Zodoende is er al heel wat boekhoudwerk gedaan. Om nu echter statistische operaties op de output van de SCANNER te kunnen verrichten moet eerst nog een programma worden aangeroepen dat de woorden (of welke andere eenheid ook in de output file) telt.

Een telling van de gelijke elementen in die lijst doet de computer eenvoudig door de woorden paarsgewijs te vergelijken en bij gelijkheid van de twee een teller op te hogen. Bij ongelijkheid maakt hij een nieuwe teller aan voor de vergelijking met het weer volgende woord in de lijst. Kortom, de statistische operaties kunnen beginnen, tenminste wanneer men nu al op deze uitsluitend uit de fysieke structuur afgeleide informatie statistiek wil gaan bedrijven.

```
BOOTEXT:
      PROC OPTIONS(MAIN);
      DCL POR CHAR(1) DEF CARD POS(1),
            NULL BUILTIN,
       CARD CHAR(80) STATIC,
       LTT CHAR(1),
       CARDA(80) CHAR(1) DEF CARD UNALIGNED,
      WO CHAR(20) STATIC,
        1 IMAGE BASED(S),
      2 WORD CHAR(20),
         2 ((LANG,RG) FIXED BIN),
         2 SOORT CHAR(5),
   (IA,A1,LB,KAP,PAG,IZI) FIXED BIN STATIC,
   (TEX,REG) FIXED BIN STATIC,
         (A,B,E) FIXED BIN STATIC,
      F5 FIXED BIN STATIC,
          N FIXED BIN STATIC,
     (TO1,ISO) FIXED BIN STATIC,
          ZI(100) PTR STATIC,
          A2 (CHAR(1),
   VULOP CHAR(5) STATIC;
      ON ENDFILE(DAFDU) GOTO UIT;
          ALLA:
               PROCEDURE;
                        ALLOCATE IMAGE;
                        IF E=0    THEN;
                                  ELSE
                        WO=SUBSTR(CARD,B,E);
                          IMAGE.SOORT=' ';
                          L=INDEX(WO,'*');  IF L=1 THEN DO;
                               E=E-1;   WO=SUBSTR(WO,2,E);
                               IMAGE.SOORT='SUBST';   ISO=0;
                                                      END;
                          IMAGE.WORD=WO;
                        TO1=TO1+1;
                        IMAGE.LANG=E;  IMAGE.RG=REG;
                        E=0;       B=N+1;
                        A=A+1;   ZI(A)=S;
               RETURN;
          END ALLA;
          OLLA:
               PROCEDURE;
           WO=' ';    ISO=1;
                    RETURN;
          END OLLA;
VULIN: PROC;
      DCL PK PTR;
            IZI=IZI+1;
                        DO J= 1 TO A;
                           PK=ZI(J);
                PUT SKIP FILE(CELAN) EDIT(PK->IMAGE.WORD,
                   PK->IMAGE.SOORT,A1,A2,LB,KAP,PAG,PK->IMAGE.RG,
                   PK->IMAGE.LANG,IZI,J,IA)
                        (A,A,F(1),A,F(1),4(F(2)),F(5),F(2),F(5));
                        FREE PK->IMAGE;
        PK=NULL;
                        END;
           IF F5=0 THEN IA=IA+1;
              F5=0;
                        DO J=1 TO A;   ZI(J)=NULL;   END;
                        A=0;
END VULIN;
ALLO: PROC;
                        CALL ALLA;
       IF ISO=1 THEN DO;
            IMAGE.SOORT=VULOP;   ISO=0;   VULOP=' ';
              END;
      CALL VULIN;
END ALLO;
PP: PROC;

      IF E>0 THEN CALL ALLA;
CALL OLLA;
      CALL ALLO;
END PP;
           OPEN FILE(CELAN) STREAM OUTPUT LINESIZE(50),
                FILE(DAFDU) STREAM  INPUT LINESIZE(80);
            A1=1;   A2='.';   LB=1;   KAP,PAG,TEX=0;   REG=0;
         IZI,ISO=0;     A=0;    F5=0;
           WO=' ';        TO1=0;
           DO 1= 1 TO 100;
              ZI(I)=NULL;
           END;
             CALL INFOTEX;
      IA=1;
```

```
   IN:
         GET FILE(DAFDU) EDIT(CARD)  (A(80));
         IF POR='@' THEN DO;
               /*  NIEUWE TEKST  */
             REG=0;   TEX=TEX+1;
                      GOTO IN;
                       END;
         IF POR=':' THEN DO;
                       /* NIEUWE PAGINA  */
             REG=0;   PAG=PAG+1;   GOTO IN;
                      END;
           IF POR='#' THEN DO;
                /*  HOOFDSTUK  */
              KAP=KAP+1;   GOTO IN;
                      END;
          IF CARD=' ' THEN DO;
           /* ANDER BOEK */
              LB=LB+1;   PAG,REG,TEX=0;
                       END;
      VRD:
           NPP=INDEX(CARD,'   ');
           IF NPP=1 THEN GOTO IN;
           IF NPP=0 THEN NPP=80;
           B=1;   E=0;
              ISO=0;
           IF CARDA(80)=' ' THEN
           REG=REG+1;
           DO N=1 TO NPP;
                 LTT=CARDA(N);   L=0;
                 IF LTT >='A' & LTT <='Z' THEN DO;
                      IF LTT ^='!' THEN L=1;
                                                 END;
                 IF LTT >='0' & LTT <='9' THEN L=1;
                 IF LTT ='*' THEN L=1;
                 IF L=1 THEN DO:
                         E=E+1;   GOTO FLAP;
                                  END;
               IF LTT='('  THEN LTT='@';
               IF LTT=')'  THEN LTT='@';
                 IF LTT ^=' ' THEN DO;
        IF LTT=',' THEN DO;  F5=1; VULOP='KOMMA'; END;
                 IF LTT='.' THEN VULOP='PUNT';
        IF LTT=':' THEN VULOP='DUBBP';
        IF LTT='"' THEN VULOP='QUOTE';
        IF LTT='@' THEN VULOP='HAAKI';
        IF LTT='?' THEN VULOP='VRAAG';
            IF LTT='!' THEN VULOP='UITRO';
           IF LTT='''' THEN VULOP='QUOTE';
        IF LTT=';' THEN VULOP='PTKOM';
                 IF LTT='-' THEN DO;
           /* VERSCHIL AFBREEK-STREEP EN GEDACHTENSTREEP  */
                       LTT=CARDA(N-1);   IF LTT >='A' & LTT <='Z'
                           THEN VULOP='STR';
                           ELSE VULOP='STREE';
                             END;
                 IF LTT='.' THEN DO;
                      LTT=CARDA(N-1);   IF LTT >='A' & LTT <='Z'
                      THEN DO;
                           LTT=CARDA(N-2);   IF LTT=' ' ! LTT='.' THEN
                                VULOP='AFK';
                           END;
                      ELSE VULOP='PUNT';
                                END;
          CALL PP;
                          ISO=0;  VULOP=' ';
        IF E=0 THEN GOTO FLAP;
                        END;
        IF LTT=' ' THEN DO;
           IF E>0 THEN CALL ALLA;
                ELSE B;N+1;
             END;
        FLAP:
           END;
        GORO IN;
UIT:
   IA+IA-1;
      PUT SKIP EDIT('TOTAAL AANTAL SYNTAKTISCHE RECORDS',TO1,
          'TOTAAL AANTAL ZINNEN',IA)  (A(35),F(5),A(25),F(5));
        PUT SKIP LIST('INHOUD PER RECORD');
        PUT SKIP LIST ('1:WOORDVORM,  2:SYNTAKTISCHE KLASSE,
        3:BOEK,    4:HOOFDSTUK,  5:PAGINA,  6:REGEL,
        7:LENGTE WOORD, 8:PHRASE 9:RANGORDE IN PHRASE,
        10:ZIN');
END BOOTEXT;
```

```
ACH          1.1 0 0 1 3    1 1     1
        KOMMA1.1 0 0 1 0    1 2     1
WAS          1.1 0 0 1 3    2 1     1
HET          1.1 0 0 1 3    2 2     1
MAAR         1.1 0 0 1 4    2 3     1
ZO           1.1 0 0 1 2    2 4     1
DAT          1.1 0 0 1 3    2 5     1
DE           1.1 0 0 1 2    2 6     1
MENS         1.1 0 0 1 4    2 7     1
PRECIES      1.1 0 0 2 7    2 8     1
WIST         1.1 0 0 2 4    2 9     1
WAT          1.1 0 0 2 3    210     1
HIJ          1.1 0 0 2 3    211     1
DEED         1.1 0 0 2 4    212     1
        KOMMA1.1 0 0 2 0    213     1
ALS          1.1 0 0 2 3    3 1     1
HIJ          1.1 0 0 2 3    3 2     1
LAS          1.1 0 0 2 3    3 3     1
        PTKOM1.1 0 0 2 0    3 4     1
WE           1.1 0 0 3 2    4 1     2
ZOUDEN       1.1 0 0 3 6    4 2     2
ONS          1.1 0 0 3 3    4 3     2
DAN          1.1 0 0 3 3    4 4     2
DE           1.1 0 0 3 2    4 5     2
MOEIZAME     1.1 0 0 3 8    4 6     2
LEKTUUR      1.1 0 0 3 7    4 7     2
VAN          1.1 0 0 3 3    4 8     2
DIT          1.1 0 0 3 3    4 9     2
BOEK         1.1 0 0 3 4    410     2
KUNNEN       1.1 0 0 3 6    411     2
BESPAREN     1.1 0 0 4 8    412     2
        PUNT 1.1 0 0 4 0    413     2
```

2.1.1 Programmatuur voor statistisch onderzoek van tekstmateriaal; een korte inleiding in SPSS

Doel is wegwijs worden in eenvoudige toepassingsmogelijkheden van SPSS.

Wat betekent SPSS? SPSS is de afkorting van Statistical Package for the Social Sciences, een computerprogramma voor de sociale wetenschappen, een programma dat gebruikt wordt voor de verwerking van statistische gegevens.

Voor SPSS-gebruik kan men het handboek (= manual) raadplegen. De titel van het manual is: SPSS: statistical package for the social sciences. De auteurs zijn N.H. Nie, C.H. Hull, J.G. Jenkins, K. Steinbrenner, D.H. Brent. Het boek is uitgegeven door McGraw-Hill, New York, 1975 (2nd edition).

De commando's die men de computer moet geven om SPSS uit te voeren zijn tweeledig: we beginnen iedere regel met een commando, daarna gaan we naar de 16e positie van die regel, waar dit commando nader wordt gespecificeerd.

Het eenvoudigste programma in SPSS is 'frequencies'. Het commando frequencies laat frequentieverdelingen zien. We zullen nu een programma met 'frequencies' maken dat woorden in een tekst telt.

We beginnen in SPSS met 'run name'. Dit is het commando dat in de computer wordt ingevoerd om de naam van de analyse aan te geven. We krijgen dan 'run name'; tussen run en name een spatie; daarna slaan we zoveel posities van de regel over tot we op de 16e positie zijn. Dan kunnen we 'kenmerken van teksten' intypen. In de output komt dan te staan 'kenmerken van teksten'. We weten dan dat de berekeningen die de computer geeft op tekst betrekking hebben. Natuurlijk mag iedere andere 'run name' ook.

Voorbeeld:

run name _ _ _ _ _ _ _ kenmerken van teksten

Het volgende commando na 'run name' is 'variable list'. Dit commando wordt op de 16e positie van de regel gevolgd door een opsomming van de variabelen. Een variabele is een naam voor de analyse-eenheid. In het onderhavige geval willen we woorden tellen, dus de naam die we kiezen luidt 'woorden'. Deze naam komt dus in de 16e kolom van de regel 'variable list', als volgt:

variable _ list _ _ woorden

Nu vraagt het SPSS-programma om te vertellen waar de woorden staan. Teksten zullen meestal aangeboden worden zoals dit boek: op iedere regel een wisselend aantal woorden; daarom is het voor tekstanalyse met SPSS handig om op de 'input format' regel in kolom 16 het woord 'freefield' te gebruiken. De woorden behoeven dan niet op vaste (= fixed) plaatsen per regel te staan. Op zichzelf is het vervelend dat ieder woord dat door SPSS bekeken gaat worden tussen aanhalingstekens (= quotation marks) moet staan. Dat laatste kan de SCANNER als extraatje opleveren. In sommige versies van SPSS kunnen bovendien geen woorden behandeld worden die langer zijn dan vier letters. De SCANNER kan hierop ook inspelen.

Er komt nu:

input _ format _ _ _ freefield

Het aardige van freefield format is dat de woorden van een tekst die verwerkt moet worden niet op vaste posities hoeven te staan. Het enige wat SPSS vraagt is dat er per variabele e v e n v e e l woorden gegeven worden.

Het eerstvolgende SPSS-commando betreft het aantal analyse-gevallen (= cases) in een tekst. Dit aantal is meestal onbekend en betreft bij tekstanalyse het aantal woorden.

```
n_of_cases_____unknown
```

Het tot nu toe geprepareerde SPSS-programma luidt volledig:

```
run_name_______kenmerken van teksten
variable_list__woorden
input_format___freefield
n_of_cases_____unknown
```

Het SPSS-pakket vraagt er nu om dat de te lezen woorden naar getallen worden gehercodeerd. Daarna kan het bedrijven van statistiek beginnen. De SPSS-terminologie voor hercoderen is 'recode'.
Als voorbeeld nemen we een tekst die straks verwerkt gaat worden:

"Wie aan Celeste vraagt hoe ze haar veeleisende werk met haar huishouden combineert, loopt kans op een merkwaardig antwoord: Precies als een ongehuwde vrouw - of man - want die hebben ook een huishouding. En aansluitend zegt ze: er is op het gebied van maatschappelijke veranderingen zoveel gepresteerd door de ongehuwden. We moeten ons niet zo doodstaren op die gehuwde vrouwen."

Door de SCANNER laten we ieder woord tussen accenten plaatsen en de scheidingstekens zoals de punten, dubbele punten, streepjes, etc. van te voren weghalen. De gewenste hercodering betreft de volgende twaalf woorden:

wie, vraagt, hoe, ze, haar, werk, met, huishouden, combineert, als, een, ongehuwde

Alle andere woorden krijgen de code 13.

In SPSS gaat dat als volgt:

```
recode__________woorden   ("wie"          = 1)
                          ("vraagt"       = 2)
                          ("hoe"          = 3)
                          ("ze"           = 4)
                          ("haar"         = 5)
                          ("werk"         = 6)
                          ("met"          = 7)
                          ("huishouden"   = 8)
                          ("combineert"   = 9)
```

```
("als"        = 10)
("een"        = 11)
("ongehuwde"  = 12)
(else = 13)
```

Hierdoor worden alle woorden gehercodeerd tot 13 getallen.

Nu zijn we toe aan de commando's om de computer statistiek te laten bedrijven. We nemen hiervoor 'frequencies'. We krijgen op de output een frequentieverdeling te zien. Op de 16e positie komt 'general=(met daarachter de variabelen-naam waarover de frequentieverdeling gemaakt moet worden)', dus 'general=woorden':

```
frequencies        general=woorden
```

Hierna komt het commando 'read input data'. In de 16e kolom komt niets te staan. Dit is de opdracht aan de computer om de gegevens in te lezen:

```
read input data
```

Op de volgende regels geven we nu de data. Daarna als laatste commando:

```
finish
```

Onze opdracht aan de computer komt er nu zo uit te zien:

```
RUN NAME        KENMERKEN VAN TEKSTEN
VARIABLE LIST   WOORDEN
INPUT FORMAT    FREEFIELD
N OF CASES      UNKNOWN
RECODE          WOORDEN("WIE"        = 1)
                       ("VRAAGT"     = 2)
                       ("HOE"        = 3)
                       ("ZE"         = 4)
                       ("HAAR"       = 5)
                       ("WERK"       = 6)
                       ("MET"        = 7)
                       ("HUISHOUDEN"  = 8)
                       ("COMBINEERT"  = 9)
                       ("ALS"        =10)
                       ("EEN"        =11)
                       ("ONGEHUWDE"  =12)
                       (else         =13)
```

```
FREQUENCIES    GENERAL=WOORDEN

READ INPUT DATA
- hier staan de data -
end-of-record commando
FINISH
```

(Let op de spaties en komma's en geen punten neerzetten achter de zinnen (in de SPSS-commando's).)
De computer zal op de output na deze opdracht een frequentieverdeling van alle woorden laten zien:

WOORDEN

CATEGORY LABEL	CODE	ABSOLUTE FREQ	RELATIVE FREQ (PCT)	CUM FREQ (PCT)
	1.	1	1.7	1.7
	2.	1	1.7	3.3
	3.	1	1.7	5.0
	4.	2	3.3	8.3
	5.	2	3.3	11.7
	6.	1	1.7	13.3
	7.	1	1.7	15.0
	8.	2	3.3	18.3
	9.	1	1.7	20.0
	10.	1	1.7	21.7
	11.	3	5.0	26.7
	12.	2	3.3	30.0
	13.	42	70.0	100.0
		------	------	
	TOTAL	60	100.0	

Uit deze tabel valt af te lezen dat het woord "wie" (met code 1) eenmaal en het woord "een" (met code 11) driemaal voorkomt. De "overige" (else) woorden (met code 13) maken 70% uit van het gehele bestand, in totaal 42 van de 60 woorden uit deze tekst, etc. Zo telt SPSS de woorden uit een tekst.

HISTOGRAM

Met behulp van het SPSS-programma 'frequencies' kunnen we een staafdiagram laten afdrukken. Een staafdiagram noemt men meestal 'histogram'. In SPSS is het alleen mogelijk een histogram te krijgen van de absolute frequencies. Deze absolute frequencies staan in de eerste kolom van 'frequencies'. Een histogram van de relatieve frequentieverdeling is met SPSS niet mogelijk. Een histogram krijgen we door de opdracht

```
options          8
```

Options is een opdracht, die altijd na frequencies komt en voor statistics. Het komt in deze volgorde:

```
frequencies      all
options           8
```

```
WOORDEN

     CODE
          I
       1. **  (      1)
          I
          I
       2. **  (      1)
          I
          I
       3. **  (      1)
          I
          I
       4. ***  (      2)
          I
          I
       5. ***  (      2)
          I
          I
       6. **  (      1)
          I
          I
       7. **  (      1)
          I
          I
       8. ***  (      2)
          I
          I
       9. **  (      1)
          I
          I
      10. **  (      1)
          I
          I
      11. ****  (      3)
          I
          I
      12. ***  (      2)
          I
          I
      13. ****************************************  (     41)
          I
          I
          I.........I.........I.........I.........I.........I
          0         10        20        30        40        50
          FREQUENCY
```

2.2 EEN LIJSTENBRIJ VAN TEKSTEN: BOEKHOUDEN 2

Als men alle fietsen van Nederland achter elkaar zou plaatsen, zou men een rij kunnen vormen van Amsterdam tot Kabul. Berekent men nu, hoeveel woorden men achter elkaar moet plaatsen om de lengte van een fiets te hebben, dan weet men ook of men Goethe's Faust, Shakespeare's Hamlet of W. F. Hermans' Onder Professoren nodig heeft om eveneens in Kabul te kunnen belanden. Een eenvoudig programma voor boekhouden ziet er bijvoorbeeld als volgt uit:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. DAG-VERKOOP.
AUTHOR. BOOT.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. CYBER-175.
OBJECT-COMPUTER. CYBER-175.
SPECIAL-NAMES.
DECIMAL-POINT IS COMMA.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT VERKOOP-FILE ASSIGN TO INPUT.
    SELECT PRINT-FILE ASSIGN TO OUTPUT.

DATA DIVISION.
FILE SECTION.
FD  VERKOOP-FILE
    LABEL RECORDS ARE OMITTED,
    DATA RECORD IS VERKOOP-REC.
01  VERKOOP-REC.
    02 AFDELING PIC X(9).
    02 FILLER PIC X(1).
    02 BEDRAG PIC 9(6)V99.
    02 FILLER PIC X(63).
FD  PRINT-FILE
    LABEL RECORDS ARE OMITTED,
    DATA RECORD IS REPORT-REC.
01 REPORT-REC PIC X(80).

WORKING-STORAGE SECTION.
77  TOTAAL PIC 9(6)V99 VALUE ZEROS.
77 AANTAL PIC 9 VALUE ZERO.
01  HEADING-0.
    02 FILLER PIC X(26) VALUE SPACES.
    02 FILLER PIC X(24) VALUE "DAGELIJKS VERKOOP REPORT".
01  HEADING-1.
    02 FILLER PIC X(16) VALUE SPACES.
    02 FILLER PIC X(8) VALUE "AFDELING".
    02 FILLER PIC X(10) VALUE SPACES.
    02 FILLER PIC X(7) VALUE "VERKOOP".
01 PRINT-LINE.
    02 FILLER PIC X(16) VALUE SPACES.
    02 AFD PIC X(9).
    02 FILLER PIC X(9) VALUE SPACES.
    02 BEDR PIC 9.999.999,99.
01  TOTAL-LINE.
    02 FILLER PIC X(16) VALUE SPACES.
    02 FILLER PIC X(6) VALUE "TOTAAL".
    02 FILLER PIC X(12) VALUE SPACES.
    02 TOT-BEDRAG PIC 9.999.999,99.
01 GEMIDDELDE-LINE.
    02 FILLER PIC X(16) VALUE SPACES.
    02 FILLER PIC X(10) VALUE "GEMIDDELDE".
    02 FILLER PIC X(8) VALUE SPACES.
    02 GEM-BEDRAG PIC 9.999.999,99.
```

```
PROCEDURE DIVISION.
BEGIN.
    OPEN INPUT VERKOOP-FILE OUTPUT PRINT-FILE.
HEADING-ROUTINE.
    WRITE REPORT-REC FROM HEADING-0 AFTER 5 LINES.
    WRITE REPORT-REC FROM HEADING-1 AFTER ADVANCING 3 LINES.
    MOVE SPACES TO REPORT-REC.
    WRITE REPORT-REC AFTER ADVANCING 3 LINES.
PROGRAM-ROUTINE.
    READ VERKOOP-FILE AT END GO TO EIND.
    ADD 1 TO AANTAL.
    MOVE AFDELING TO AFD.
    MOVE BEDRAG TO BEDR.
    ADD BEDRAG TO TOTAAL.
    WRITE REPORT-REC FROM PRINT-LINE AFTER ADVANCING 1 LINES.
    GO TO PROGRAM-ROUTINE.
EIND.
    DIVIDE AANTAL INTO TOTAAL GIVING GEM-BEDRAG ROUNDED.
    MOVE TOTAAL TO TOT-BEDRAG.
    WRITE REPORT-REC FROM TOTAL-LINE AFTER ADVANCING 2 LINES.
    WRITE REPORT-REC FROM GEMIDDELDE-LINE AFTER 1 LINES.
    CLOSE VERKOOP-FILE PRINT-FILE.
    STOP RUN.
```

De output van dit programma ziet er als volgt uit:

```
          DAGELIJKS VERKOOP REPORT

AFDELING          VERKOOP

FILIAAL-1         0.009.855,0
FILIAAL-2         0.021.457,5
FILIAAL-3         0.091.234,5

TOTAAL            0.122.547,36
GEMIDDELDE        0.040.849,12
```

De input was:

```
FILIAAL-1 0098550
FILIAAL-2 0214575
FILIAAL-3 0912345
```

Bij boekhouden telt het resultaat, het harde cijfer. De realiteitswaarde van dat harde cijfer is een ander probleem. Aan teksten is heel wat boek te houden. We vonden al een schat van boekhoudkundige waarde in de informatie die de SCANNER opleverde. Er wordt echter meer boek gehouden in teksten. Van teksten worden indexen, concordanties en woordenboeken samengesteld. Indexen en concordanties worden tegenwoordig geregeld met standaardprogrammatuur. Deze standaardprogrammatuur is in een dergelijke staat dat het weinig zinvol is, de technieken voor het schrijven van programma's voor beide taken uitgebreid aan de orde te stellen. Ook teksteditoren beschikken meestal over functies die indexen en concordanties kunnen maken. Programma's om woordenboeken te maken zijn er ook in overvloed maar toch nog niet in een dergelijke status dat ze zomaar kunnen worden opgeroepen. Het woordenboek-probleem grijpen we dan ook aan om de eerste basis-kennismaking met de hoofdprogrammeertaal (LISP) van onze tekstmachine te doen verlopen. Daarvoor echter eerst Indexen en concordanties.

2.3 INDEXEN EN CONCORDANTIES

Onder indexen verstaan we lijsten die bestaan uit alle woorden die voorkomen in teksten. Deze woorden worden slechts eenmaal afgedrukt, wanneer ze meer dan eens in de tekst voorkomen.

Onder concordanties verstaan we hetzelfde als onder indexen, met dien verstande dat nu behalve het woord zelf ook een gedeelte van de omgevende woorden (context) wordt afgedrukt. Komt een woord meer dan eens voor, ook dan weer is het opnieuw met context afgedrukt.

Deze lijsten zijn nuttig voor verschillende soorten van onderzoek. Men kan erdoor een eerste indruk krijgen van de woordenschat van een tekst (index) en van de zinsstructuren in een tekst (concordantie). Deze lijsten worden dan ook dikwijls gebruikt als voorbereidende fase van een beschrijvend taalkundig of letterkundig onderzoek. Aangezien veel beschrijvend, met name letterkundig onderzoek wordt gewijd aan vocabulaire studie zijn deze instrumenten van redelijk belang. Zeker wanneer men bedenkt dat in het verleden en eigenlijk nog in het heden veel van dit soort onderzoek met de hand wordt gedaan: een hooggeleerde of laaggeleerde schrijft dan één voor één alle woorden op die hij tegenkomt in het werk van een bepaalde schrijver, of bij dialectonderzoek in de spraak van een geïnterviewde, alles netjes op kaartjes met vindplaats etc. De eerste moeilijkheid is dan de enorme hoeveelheid woorden die teksten al gauw bevatten. De tweede pas echt onaangename moeilijkheid is dat men de woorden op meer dan één manier zou willen sorteren: oplopend in frequentie, aflopend in frequentie, in omkering, dat is de woorden van voor naar achter (retrograad: zie paragraaf 2.4), enz.

Dit soort onderzoek is in feite onmogelijk zonder computerprogramma's die de boekhouding verrichten. Het blijkt nu dan ook dat inzet van de computer alle resultaten uit de literatuur omverwerpt. Een aardig voorbeeld is de studie naar het woordgebruik van de Russische dichter Ossip Mandel'stam dat in Utrecht werd uitgevoerd. Bij dat onderzoek werd gebruik gemaakt van het indexprogramma uit de bibliotheek LIBOLIB dat zich in de appendix bij dit hoofdstuk bevindt. Dit programma is weliswaar niet zo sophisticated als het programma waaraan dit hoofdstuk verder wordt gewijd, maar inzet ervan leverde toch al snel op dat de uitspraken in de literatuur over Mandel'stam alle moesten worden herzien. Er is aardig wat geschreven over de vier perioden waarin Mandel'stams werk wordt verdeeld en over de woorden die hij bij voorkeur in die verschillende perioden zou hebben gebruikt. Ze bleken niet te vinden in de indexen die de computer had gemaakt. Toch was het programma betrouwbaar en liet niet op eigen gelegenheid woorden weg om de mens een poets te bakken.

2.3.1 Het programma OCP

Veel programma's zijn in de loop van de geschiedenis van de computerlinguistiek vervaardigd, stilistische analyses gedaan, onderzoek naar anonieme auteurs, ontwerp van taalleermateriaal en de studie van rijmschema's kon worden ondernomen. Voor historisch onderzoek was de meest voor de hand liggende toepassing van 'lijstenbrij-programma's' een sorteerprogramma dat de tekst in alfabetische of frequentievolgorde plaatste of het opzoeken van gedeelten in de tekst die aan speciale voorwaarden voldeden, bijvoorbeeld het zoeken van mensen met dezelfde achternaam in een geboorteregister, om zodoende de gang door de geschiedenis van een familie te vinden.

Het meest beroemde programma waardoor dit soort taken kon worden verricht was COCOA, zo genoemd naar de afkorting van 'A word COunt COncordance' (de A achterop geplaatst). Het programma werd ontwikkeld in een samenwerkingsproject tussen het Atlas Computer Lab in Berkshire en de universiteit van Cardiff. Het werd gepubliceerd in april 1973. Met dit programma kan men woordtellingen verrichten, concordanties maken, woordfrequentieprofielen maken. Men kan er echter ook interessantere tellingen door laten verrichten. Stel dat men Goethe's Faust heeft verponst zodat de computer in staat is de tekst te lezen, en dat men ook precies heeft aangegeven wanneer wie wat zegt, dan kan men met COCOA alleen die woorden tellen (etc.) die Faust zelf zegt, of de woorden van Gretchen. Zo kan men dus een vrij genuanceerde boekhouding van een bestand (= data base) krijgen. Ook kan men commentaar in zijn tekst laten opnemen, bijvoorbeeld regieaanwijzingen, zonder dat ze ook in de index terecht komen. Deze commentaarteksten zouden immers het beeld van het geheel vertroebelen als ze wel in de index kwamen.

Op zich is COCOA dus een vrij krachtig programma. Het levert bovendien nog output files op die regelrecht op SPSS kunnen worden aangesloten. Het nadeel van COCOA is echter dat het vrij veel inspanning kost om ermee te leren werken: COCOA is niet zo erg gebruikersvriendelijk, zoals dat met een modern woord heet.

Andere woord-lijstenprogramma's zoals bijvoorbeeld JEUDEMO, CLOC, VERA, FAMULUS, FIND2 waren niet zo algemeen als COCOA en daardoor geen alternatief. Ook moet men dikwijls eerst andere zaken gaan leren, zoals bijvoorbeeld bij FAMULUS, een programma om informatie op te halen (= information retrieval) en opslaan/catalogiseren. Om FAMULUS te kunnen gebruiken moet men eerst leren programma's schrijven in een andere programmeertaal, nl. FORTRAN of COBOL, en bovendien moet men op de hoogte zijn van concepten en begrippen uit een ander programma, nl. CODASYL, een programma voor het onderhouden van databestanden (een 'Data Base Management System', of DBMS).

Al deze ontwikkelingen leidden er in 1977 toe dat men er in Oxford over ging denken een vervanging van COCOA te maken, waar-

in de sterke kanten van COCOA en andere programma's bewaard bleven en de zwakke verdwenen zouden zijn. Het programma werd genoemd OCP (Oxford Concordance Project) en het moest volledig onafhankelijk zijn van de machine waarop het ooit zou worden ingebracht (= geïmplementeerd). Ook moesten alle wensen van potentiële gebruikers in het ontwerp van het programma worden meegenomen. Om die machine-onafhankelijkheid te verkrijgen werd OCP geschreven in FORTRAN en tegelijkertijd getest en ontwikkeld op vijf verschillende machines, nl.: ICL 1096A, ICL 2980, CDC 7600, IBM 370/168 en een DEC 10. Om alle gebruikerswensen te achterhalen werden alle universiteiten in Groot Brittanië bezocht.

Werking en gebruik van OCP

Typt men het volgende in:

*GO

dan krijgt men de volgende gegevens over zijn databestand:

1 - een alfabetische woordindex van alle woorden in de tekst (alfabetisch in de volgorde van het Latijnse of liever Engelse alfabet);
- getallen uit de tekst worden geprint voorafgaand aan de woorden;
- de maximale woordlengte is 20 letters;
- de frequenties van alle woorden en getallen worden geprint;
- de maximale frequentie is 99.999.

De voorwaarden waaronder *GO deze prestatie levert zijn:

- de maximale woordlengte is 20 letters;
- de tekst is verdeeld in eenheden van 80 tekens (een kaartbeeld dus);
- de tekst mag maximaal 10.000.000 regels bevatten.

2 Men krijgt echter ook een concordantie van dezelfde tekst. De concordantie heeft dezelfde specificaties als de index met dien verstande dat de context bestaat uit 120 tekens en dat per pagina 60 regels van 120 tekens verschijnen.

Dat is wel uiterst simpel. Men kan dan ook vaststellen dat wat dit aspect van OCP betreft, het criterium van gebruikersvriendelijkheid door de ontwerpers gehaald is.

OPDRACHTENTAAL (COMMAND LANGUAGE) VAN OCP

Met de opdracht heeft men gebruik gemaakt van de opdrachtentaal (Command language) van OCP. Deze opdrachtentaal is onderverdeeld in SECTIES, OPDRACHTEN en ELEMENTEN.

SECTIES

Secties worden gekenmerkt door sleutelwoorden (= key words), d.w.z. vaste woorden die worden voorafgegaan door een asterisk. Sectiehoofden zijn de woorden '*INPUT', '*WORDS', '*ACTION', '*FORMAT'. Een sectie eindigt wanneer een nieuw sectiehoofd verschijnt. Het eind van alle secties wordt gekenmerkt door het sleutelwoord *GO.

Uit het voorbeeld dat werd gegeven met de simpele commando *GO kan men de conclusie trekken dat veel operaties standaard (= default) zijn. Immers, wanneer men slechts het eind van een sectie opgeeft met *GO en anders niets, dan gebeurt er toch veel tekstverwerking.

- Formele definitie van de opdrachtentaal van OCP

```
<COMMAND-LANGUAGE>    ::= <SECTION><COMMAND> <ELEMENT>
<SECTION>             ::= <SECTION-HEADER> <END>
<SECTION-HEADER>      ::= ['*INPUT', '*WORDS', '*ACTION,
                           '*FORMAT']
<END>                 ::= ['*GO']
```

- Commentaar bij de formele definitie

De formele definitie die boven werd gegeven is geschied in de zogenaamde BNF notatie. Deze manier van schrijven ter definitie van talen werd ontwikkeld door Backus en Naur ten behoeve van een doorzichtiger formulering van programmeertalen, vandaar de naam Backus Naur Format. Het is een toepassing van de zogenaamde 'contextvrije' grammatica. Een contextvrije grammatica werkt als volgt: men begint bij het meest algemene begrip en daalt dan stapsgewijs af naar de meest specifieke eenheden. Verder verdeelt men de definitie in twee helften: het gedeelte links van de 'definitie-operator' (het teken '::=') en het gedeelte rechts van de definitie-operator. Links van de definitie-operator mag slechts één begrip of woord voorkomen. Dit woord kan worden vervangen door alle woorden die rechts van de definitie-operator staan. Tussen spitse haken staan woorden die nog verder moeten worden gedefinieerd (zogenaamde niet-terminale symbol en of variabelen - ze vormen geen eindformulering). Tussen vierkante haken staan verzamelingen van woorden; tussen aanhalingstekens strings of literals. Woorden tussen aanhalingstekens moeten

letterlijk zo worden gebruikt (vandaar de naam literals). De woorden tussen aanhalingstekens zijn ook niet verder definieerbaar; het zijn de eindsymbolen of terminale symbolen. Het teken '-' wordt gebruikt als 'tot operator', d.w.z. 'het loopt tot'. Deze operator is niet standaard BNF. In veel technische literatuur van enig niveau vindt men een dergelijke wijze van definiëren van hetzij de data base, hetzij de programmeertaal, hetzij het programma, hetzij alle drie.

Uit het bovenstaande mag duidelijk zijn geworden dat deze wijze van noteren zeer doorzichtig en helder is en inderdaad allerlei misverstanden die in gewoon taalgebruik zitten, voorkomt.

Om tot een volledige definitie van OCP te komen moeten nu nog de secties *INPUT, *WORDS, *ACTION en *FORMAT worden gedefinieerd. Daarna kunnen programma's in OCP worden geschreven.

*INPUT

De opdrachten in deze sectie geven aan hoe de tekst van de data base werd getypt. Het programma OCP neemt altijd kaartbeelden als input, dus 80 kolommen. Men kan ermee specificeren van waar tot waar de tekst loopt waarop gewerkt moet worden, bijvoorbeeld van kolom 5 tot 75. Verder kan men aangeven waar een nieuwe regel begint. Heeft men bijvoorbeeld input-tekst met korte regels bijvoorbeeld van slechts 20 letters, dan kan men na iedere regel een slash / schrijven en gewoon doortypen op die regel. Hiermee spaart men vanzelfsprekend papier uit als er geponst wordt. Wel kan men in moeilijkheden komen bij het verbeteren, wanneer men geen tekst-editor gebruikt, maar handwerk verricht. Ook kan het zijn dat een tekstregel meer dan 80 kolommen bevat, zodat er een voorziening moet worden getroffen op grond waarvan het programma het volgende kaartbeeld leest. Ook dat kan, en wel met de reservering van een speciaal teken, bijvoorbeeld '+'. Tot slot kan men nog opgeven welke tekst niet moet worden gelezen. Zo kan het voorkomen dat midden in een tekst commentaar voorkomt, of zoals bij toneelstukken regie-aanwijzingen. Stel dat men die tussen dubbele ronde haken '((' en '))' heeft ingebracht, dan kunnen ook deze tekens worden gebruikt bij de definitie van de tekst.

- Reservering van bijzondere tekens in de input

1 Kaartposities worden aangegeven met de opdracht TEXT TO, waarbij zowel links van TO een getal staat (DEFAULT is 1) als rechts. In ons voorbeeld dus

```
TEXT 5 TO 75;
```

2 Nieuwe regel in lopend record wordt aangegeven met de opdracht NEWLINE = ... Daar we de slash hadden aangenomen wordt dit

voor ons voorbeeld dus:

NEWLINE = / .

3 Doorlopen van de regel wordt opgegeven met de opdracht CONTINUE = opgegeven character. We hadden een '+' aangenomen als teken, dus in ons geval luidt de opdracht:

CONTINUE = +.

4 Commentaar in de tekst wordt gekenmerkt door de opdracht COMMENT = " " AND " ". Tussen de aanhalingstekens staat het gebruikte tekenpaar. In ons voorbeeld waren dat 2x het ronde hakenpaar, dus de opdracht wordt:

COMMENT "((" AND "))".

- Reservering van bijzondere informatie in de input record

Het kan zijn dat men bij het inbrengen van de input-tekst vaste posities heeft gebruikt voor bijvoorbeeld paginering volgens de oorspronkelijke uitgave van de tekst of dat men bijvoorbeeld van een gedichtenbundel de versnummers heeft meegenomen en gecodeerd op een vaste positie. Stel, men heeft het paginanummer van kolom 80 tot 85 en het versnummer van kolom 86-90, dan kan men die vastleggen met de volgende opdracht:

REFERENCES 80 TO 85 = "PAGE", 86 TO 90 = V.

- Selectie van delen uit de tekst waarop een bewerking moet worden verricht

Stel dat men als data-bestand Goethe's Faust heeft ingevoerd, en dat men daarvan alleen de eerste 500 regels wil bewerken tot index om een snel overzicht te hebben van de rijkdom van het vocabulaire dat daarin wordt gebruikt, dan schrijft men de volgende opdracht:

PROCESS TO 500.

Wil men alleen een overzicht van de passages waarin Mefisto aan het woord is, dan schrijft men:

PROCESS FROM 1 TO 500, WHERE S = "MEFISTO".

Nu kan men dus het vocabulaire van Mefisto vergelijken met het algemeen gebruikte vocabulaire in de hele passage. Wil men onderzoeken of het vocabulaire van Mefisto substantieel verandert in de rest van de tekst in vergelijking met de eerste 500 regels, dan schrijft men:

PROCESS TO 500; WHERE S = 'MEFISTO".

Stel dat men nu vanaf 500 alle informatie wil hebben, maar niet de informatie over de tekst tussen 525 en 552 omdat men weet dat daar-

in niet-relevante informatie is opgenomen, dan schrijft men:

```
PROCESS FROM 501, EXCEPT FROM 525 TO 552.
```

Verdere mogelijkheden zijn: selectie op auteur,
dit geschiedt met de opdracht:

```
PROCESS WHERE A = "SARTRE"
```

(als men alle citaten uit de werken van Sartre in het data-bestand wil hebben); en selectie op kenmerk:
stel dat men geen directe uitspraken van Sartre wil hebben, dan schrijft men:

```
PROCESS WHERE A = "SARTRE", EXCEPT BETWEEN
                  QUOTE.
```

(de directe rede staat immers tussen aanhalingstekens).
Een ander voorbeeld van selectie op kenmerk is bijvoorbeeld dat men van Goethe alle beginregels van alle gedichten uit het jaar 1815 uit zijn data-bestand wil ophalen. Men schrijft dan:

```
PROCESS WHERE Y = "1815", WHERE A = "GOETHE",
                  TO LINE 1.
```

- Samenvatting van *INPUT

De volgende opdrachten kan men gebruiken om posities in input records te definiëren:

```
TEXT TO
NEWLINE =
CONTINUE =
COMMENT =
```

De opdracht om bijzondere informatie in input records vast te leggen:

```
REFERENCES TO =
```

Opdrachten om selectie van delen uit de tekst waarop bewerkingen moeten worden verricht aan te brengen:

```
PROCESS FROM      TO
EXCEPT WHERE =
EXCEPT BETWEEN      AND
EXCEPT FROM      TO
```

*WORDS

Deze opdracht geeft aan hoe de woorden in de tekst werden opgebouwd. Ten eerste ten aanzien van het gebruikte ALFABET. Met de opdracht ALFABET kan men namelijk opgeven welk alfabet men gebruikt heeft

en welke volgorde van letters (d.w.z. de 'collating sequence').
Ook letters die gelijk aan elkaar zijn, bijvoorbeeld hoofd- en kleine letters kunnen worden opgegeven. Default is het Engelse alfabet met grote en kleine letters als gelijk gedefinieerd en de getallen voorop in de collating sequence.
Ook kan men alfabetten opgeven waarin afgebroken woorden voorkomen. Stel dat het afbreekteken "=" werd gebruikt en men geeft de volgende lijst:

ALFABET ":= a=A, b=B, c=C, ..., z=Z".

Dan worden de afgebroken woorden gesorteerd voor het volledige woord afgedrukt. Hoofdletters zijn gelijk aan kleine letters.

Behalve het alfabet is het ook mogelijk dat men extra tekens nodig heeft, bijvoorbeeld voor accenten (diacritische tekens dus). Om die te definiëren gebruikt men de opdracht:

DIACRITICS.

Stel dat men een Franse tekst heeft ingebracht en de accenten "^" als 1, de "´" als 2 heeft gecodeerd, dan zorgt de opdracht
DIACRITICS "1=2" ervoor dat de woorden met accenten gewoon worden meegesorteerd op de plaatsen van de woorden zonder accenten (d.w.z. men krijgt de volgorde: de dé demain en niet de demain dé).

Letters weglaten kan ook, namelijk via de opdracht:

IGNORE "a, b, c, . . . "

Deze letters worden dan niet bij welke andere bewerking dan ook betrokken.

Samentrekken van letters kan ook, en wel via de opdracht:

COMPRESS de string "....." moet worden "."

Iedere letter die niet in WORD wordt gedefinieerd wordt opgevat als woordscheidingsteken (= word delimiter).

*ACTION

Deze sectie definieert precies welke actie moet worden uitgevoerd op de in de andere secties gedefinieerde data. De opdrachten zijn:

- Ten eerste: het DO command

Na het Do command schrijft men wat er te doen is, nl. een WORDLIST, een INDEX of een CONCORDANCE te maken al dan niet met statistieken gebaseerd op woordfrequentie (opdracht: STATS).

Samengevat:

DO (WORDLIST, INDEX, CONCORDANCE) STATS

Opmerking: Van de lijst tussen ronde haken moet er één worden gekozen.

- Ten tweede: het PICK command

Via deze opdracht kan men definiëren welke woorden moeten worden uitgekozen uit de data base. In het PICK command is het ook weer mogelijk het EXCEPT command te gebruiken. Verdere definities bij het PICK command maakt men met:

RANGE

Hiermee geeft men op welke elementen uit het alfabet men gesorteerd wenst te zien (bijvoorbeeld de letters "a" t/m "n"): RANGE A TO N.

LENGTH

geeft de lengte aan van het woord/de woorden die men wil sorteren, bijvoorbeeld alle woorden van 5 letters: LENGTH 5

FREQUENCY

geeft aan tot welke frequentie men de woorden gesorteerd wil hebben, bijvoorbeeld alle woorden t/m de frequentie 2: FREQUENCY LT 3
Verder heeft men bij FREQUENCY de mogelijkheden NE (niet gelijk) en GT (groter dan).

SAMPLE

maakt een keuze volgens de verder opgegeven specificaties. Bijvoorbeeld:

PICK SAMPLE OF 1000 TOKENS, " dt", RANGE "ABEL" TO "KAIN"
EXCEPT FREQUENCY GT 20.

Dat wil zeggen: Print een keuze van 1000 woordvormen af uit de data base, woorden die 4 letters lang zijn en uitgaan op "dt" en zitten in het alfabet tussen ABEL en KAIN. Ze mogen verder niet meer dan 21 maal voorkomen.

Tot slot kan men met een asterisk een willekeurige vulling van de gezochte woorden aangeven. Bijvoorbeeld: men wil alle woorden die 5 letters hebben en die uitgaan op een "t":

PICK LENGTH 5, "*T"

• Bijzondere acties

Onder de bijzondere acties vallen het DEFINE, het STOP, het SORT KEYS, het SORT CONTEXT, het REFERENCES en het MAXIMUM CONTEXT command.

- DEFINE

Met deze opdracht kan men een klasse van woorden (of strings) samenvatten onder een aanduiding. Bijvoorbeeld:

```
DEFINE "ZIJN" = "BEN BENT IS ZIJN WAS WAREN
                 GEWEEST WAART WEZE WEZEN"

DEFINE "VOLTOOID DEELWOORD" = "GE*D"
```

Met deze opdracht kan men dus op ruwe wijze lemmatiseren, dat wil zeggen niet op woordvorm maar op woordstam de woorden sorteren.

- STOP

Hiermee kan men het programma stoppen.

- SORT KEYS

Met deze opdracht kan men definiëren op welke manier een data base gesorteerd moet worden. Men kan sorteren van links naar rechts (d.w.z. wat men normaal onder alfabetisch verstaat) (opdracht: START). Men kan echter ook van achter naar voren (zogenaamd retrograad) sorteren (opdracht: END).
Verder kan men laten sorteren op woordlengte en op frequentie, beide oplopend of aflopend. De SORT KEYS opdracht ziet er dus in zijn algemene gedaante als volgt uit:

```
SORT KEYS BY (START, END)
             (ASCENDING, DESCENDING) FREQUENCY
             (ASCENDING, DESCENDING) LENGTH
```

Geeft men niets op dan krijgt men SORT KEYS BY START.

- SORT CONTEXT

Deze opdracht is alleen van toepassing wanneer men een concordantie wil maken. Met de opdracht kan men definiëren op welke manier de context voor een gegeven woord moet worden geprint. Men kan definiëren of de context links of rechts van het woord moet staan. Ook kan men contexten opgeven naar gelang hun verwijzingen (bijvoorbeeld het regelnummer). Tenslotte kan men contexten definiëren aan leestekens (rechts of links).

Samenvattend:

SORT KEYS BY (LEFT, RIGHT) OF KEYWORD
REFERENCES
(LEFT, RIGHT) PUNCTUATION

Default wordt een context aangemaakt op de volgorde waarin woorden in een tekst voorkomen.

- REFERENCES

Tot maximaal 20 letters mag men gebruiken als referenties. Default is dat het record nummer van maximaal 8 posities wordt geprint.

- MAXIMUM CONTEXT

Deze opdracht ziet er in zijn algemene gedaante als volgt uit:

MAXIMUM CONTEXT ((LEFT, RIGHT)) n LETTERS
n WORDS
COMPLETE (string, LINE)
TO $string_1$

Dat wil zeggen: men kan zelf opgeven hoeveel letters links of rechts, of hoeveel woorden links of rechts mogen voorkomen. COMPLETE vereist een volledig voorkomen van de string die men opgeeft of van de regels in de tekst.
TO $string_1$ betekent dat de context loopt van het sleutelwoord tot het in $string_1$ opgegeven woord dat dan links of rechts van het sleutelwoord moet voorkomen.

Default is: MAXIMUM CONTEXT 120 LETTERS.

• De FORMAT ACTION

Deze opdracht specificeert hoe een index of concordantie op de printer verschijnt. De volgende opdrachten vallen onder deze action: KEYS, CONTEXT, REFERENCES, LAYOUT, PRINT, TITLES.

- KEYS

Met deze opdracht kan men bepalen waar de positie is van het woord tegenover de context (voor een concordantie) of tegenover de verwijzingen (voor een index).

KEYS (ALIGNED, CENTRE, LEFT, RIGHT), (LEFT SAME LINE, RIGHT SAME LINE)
(WITH FREQUENCY)

Men kan uit beide lijsten ieder een kiezen.

Default is voor concordantie: KEYS ALIGNED, WITH FREQUENCY
voor index: KEYS LEFT SAME LINE, WITH FREQUENCY

- CONTEXT

Hiermee definieert men het maximum aantal regels op de printer voor een speciale context en of die context volledig of samengeperst moet worden afgeprint of zelfs weggelaten moet worden.

CONTEXT SIZE n (EXCLUDE, COMPRESS) COMMENTS

Voor een concordantie is default: CONTEXT SIZE 1 (d.w.z. ieder woord krijgt een regel context mee).
Default voor index: CONTEXT SIZE 10000.

- REFERENCES

Geeft aan waar de verwijzingen staan op de pagina in verhouding tot het woord, respectievelijk de afgedrukte context.

Default is: REFERENCES LEFT, TO 1.000.000

- LAYOUT

Hiermee geeft men op hoe het geheel op papier komt te staan.

LAYOUT (n COLUMNS) (BY m WIDE) (BY l LINES)
((LEFT, RIGHT) MARGIN k
MINIMUM GAP j
i LINES BELOW (ENTRIES, KEYS)

Default voor concordanties: LAYOUT 1 COLUMN BY 120 WIDE BY 60 LINES, MARGIN 0, 1 LINE BELOW ENTRIES, 1 LINE BELOW KEYS;
Default voor index: LAYOUT 3 COLUMNS BY 30 WIDE BY 60 LINES, MARGIN 0, MINUMUM GAP 4, 1 LINE BELOW ENTRIES

- PRINT

Algemene gedaante:

PRINT ALL BUT string
string AS character
UNSEEN string

ALL BUT string betekent: niet printen de letters die in string voorkomen.
String AS character betekent: de string verschijnt op het output medium zoals wordt gedefinieerd in character.
UNSEEN wordt naar het output medium gestuurd, maar niet meegerekend bij het totaal aantal letters op een regel.

Default is: alle letters worden geprint zoals ze worden gedefinieerd in *WORDS.

- TITLES

Hiermee geeft men aan welke kop op elke pagina van de output moet worden geprint. Men kan er ook voetnoten en paginanummers mee aangeven.

TITLES (string, KEYS, PAGE n)
(LEFT CYCLE, RIGHT CYCLE, CENTRE)
(BELOW)

"string" kan de gebruiker zelf opgeven. Alles wat langer is dan de opgegeven output regel wordt afgekapt.
KEYS pakt het eerste en het laatste woord van de pagina.
LEFT, RIGHT geeft aan waar de titel op de pagina komt.
CYCLE geeft aan dat bij oneven nummers de titel links (of rechts) op de pagina komt en bij de even nummers aan de andere kant.
BELOW geeft aan dat de titel aan de onderkant van de pagina verschijnt.

Default is: geen titels, geen paginanummering en geen kop.

OPGAVEN EN VRAGEN

1. Heeft het zin de parameters LEFT en RIGHT te gebruiken bij MAXIMUM CONTEXT COMPLETE LINE?
2. Schrijf de opdracht neer voor een complete run op default waarden.
3. Voor een programma run zijn alleen nodig de eerste 50 posities van input. Verder wenst men alle defaults. Schrijf het programma in OCP.
4. Men wil met OCP een concordantie maken waarbij de woorden aan het begin van de regel staan. Verder alle defaults.
5. Maak een concordantie naar de volgende specificaties.
   1 : Alleen de posities 1 tot 65 bekijken.
   2 : Alleen de regels 50 tot 5000 waarin de auteur VONDEL wordt genoemd.
   3 : De concordantie is alfabetisch geordend volgens het Engelse alfabet.
   4 : Links van het sleutelwoord staan 2 woorden.
   5 : Rechts van het sleutelwoord staan 2 woorden.
   6 : Neem alleen woorden van minder dan 10 letters.
   7 : Alleen woorden die meer dan 2x voorkomen.
   8 : Neem alle woorden van het Nederlandse "zijn" samen.
   9 : Neem als referentie categorieën P en L tot 3 letters.

10: Druk deze referenties rechts af.
11: Maak 2 kolommen van 50 letters op elke pagina.
12: Begin met pagina nummer 52.
13: Print als titel "EEN STUKJE VONDEL".

2.3.2 Inhoudelijke indexen: Liefde met kroketten

De indexen die boven werden beschreven werken voornamelijk op de fysieke structuur van een tekst. Er is overigens wel een mechanisme in OCP aanwezig om ook woorden die bij elkaar horen samen te vatten, namelijk de functie DEFINE * "...." = " ..... ".

Nu worden indexen dikwijls gemaakt om aan het eind van een boek via trefwoorden een gemakkelijke ingang in de stof van het boek te kunnen vinden. Daarbij ligt het voor de hand dat men niet alle woordvormen pakt, maar liever een samenvattend begrip noteert. Voor positieve gedragsvormen zal men dan bijvoorbeeld de kop 'liefde' in de index opnemen, waaronder dan ook een verwijzing naar de vormen van paringsgedrag als liefdesverklaringen etc. kunnen worden samengevat. Ook zal men in een boek over de culinaire geneugten van de gemiddelde Europese dis niet alle gerechten willen opsommen, zeker niet alle ingrediënten erin, maar bijvoorbeeld een ingang 'frituur' willen hebben waar dan de kroketten onder kunnen vallen.

Het instrument dat we nu hebben (OCP dus) staat een dergelijke samenvatting inderdaad toe, zij het op een wat omslachtige manier, namelijk de manier van DEFINE. Die zou men als volgt kunnen inzetten: maak een index, zie welke begripsvelden er in de tekst voorkomen en definieer die met DEFINE.

Een wat minder omslachtige manier vereist heel wat meer denkwerk. Daarvoor zou een programma ter beschikking moeten komen dat zelf de 'woordvelden' samenstelt. Hiervoor moet echter nieuwe programmatuur ontwikkeld worden. Daarvoor moet men dus zijn gereedschap voor creatief werk ter beschikking hebben staan. Bovendien moet men zich eerst enigszins hebben ingewerkt in inhoudelijk (semantisch) werk met de computer. Het gereedschap beginnen we hier aan te bieden, het semantische werk komt in hoofdstuk 3 aan de orde.

2.4 LISP

Opgave 1 – Het maken van een retrograde woordenboek

In deze opgave gaat het om 'woorden', 'woordenboeken' en om 'taken' die woorden en woordenboeken bewerken. Daarbij zullen we ons bedienen van een universele programmeertaal voor tekstverwerking: LISP.

We pakken het volgende probleem aan: maak een RETROGRADE voor het NEDERLANDS WOORDENBOEK.

Een pagina uit een retrograde woordenboek ziet eruit als in figuur 1 op p. 54.

Dit probleem lossen we op in vijf stappen.

1. We leggen Van Dale's woordenboek in het computergeheugen vast.
2. We laten de computer alle woorden uit Van Dale opblazen tot losse letterwoorden.
3. We laten de computer alle losse letterwoorden uit Van Dale omkeren.
4. We laten de omgekeerde Van Dale van voren naar achteren op alfabet sorteren, we laten alle woorden opnieuw omkeren en noemen dit 'Retrograde Van Dale'.
5. We laten de Retrograde Van Dale afdrukken.

In het volgende zullen we deze vijf stappen, waarin het maken van een retrograde woordenboek opgesplitst kan worden, programmeren. Het programmeerwerk begint onmiddellijk.

- De eerste stap op weg naar het retrograde woordenboek

Vastleggen van gegevens met behulp van de programmeertaal LISP (van LISt Processing) gaat eenvoudigweg door de taak SETQ te geven. Iedere taak in LISP wordt door een openingshaak voorafgegaan, daarna komt de naam van de taak en vervolgens de manier waarop deze uitgevoerd dient te worden, en tenslotte de sluithaak.

Nu heet de taak of opdracht dus SETQ; deze moet uitgevoerd worden om bijvoorbeeld Van Dale's woordenboek onder die naam vast te leggen in het computergeheugen. De taak SETQ werkt dus als naamgever voor de inhoud. De opdracht SETQ gaat er straks ongeveer zo uitzien:

(SETQ naam Van Dale op de juiste inhoud)

In LISP geldt de afspraak dat woorden die gescheiden worden door spaties en/of komma's als afzonderlijke woorden meedoen. Verder is er internationaal afgesproken dat een woord nooit met een cijfer begint. Als aan deze eisen voldaan is, noemt men een woord in

SLIKKERIG
FRIKKERIG
SCHRIKKERIG
SIKKERIG
BOKKERIG
HOKKERIG
SJOKKERIG
SLOKKERIG
MOKKERIG
POKKERIG
SCHROKKERIG
SOKKERIG
STOKKERIG
STUKKERIG
JANKERIG
KANKERIG
PLANKERIG
RANKERIG
STINKERIG
BONKERIG
PRONKERIG
SMOKERIG
SPOKERIG
ROKERIG
SCHROKERIG
HARKERIG
WEDERKERIG
SNORKERIG
JEUKERIG
AANHALERIG
JOVIALERIG
DROELERIG
HUILERIG
DRUILERIG
PRUILERIG
AANSTELLERIG
BEDILLERIG
RILLERIG
GRILLERIG
PRULLERIG
SCHEMERIG
KLIEMERIG
TEMERIG
SLUIMERIG
LIJMERIG
SLIJMERIG
MIJMERIG
OPKAMMERIG
VLAMMERIG
ZWAMMERIG
ZWEMMERIG
GLIMMERIG
LOMMERIG
BROMMERIG
DROMERIG
SMERIG
KLEUMERIG
KOUKLEUMERIG
GRIENERIG

GEWICHTIGDOENERIG
DIKDOENERIG
POENERIG
ZOENERIG
DREINERIG
SPINNERIG
OERIG
BOERIG
HOERIG
POETELOERIG
MOERIG
RUMOERIG
ROERIG
LEENROERIG
OPROERIG
BREEDVOERIG
UITVOERIG
GAPERIG
SLAPERIG
SCHRAPERIG
GENIEPERIG
GNIEPERIG
PIEPERIG
GRIEPERIG
SNOEPERIG
KROEPERIG
SOEPERIG
PEPERIG
STREPERIG
ZEPERIG
GLUIPERIG
KRUIPERIG
KNIJPERIG
GRIJPERIG
KRIMPERIG
STUMPERIG
KOPERIG
LOPERIG
VERLOPERIG
STROPERIG
KNAPPERIG
PAPPERIG
AANPAPPERIG
RAPPERIG
OPSCHEPPERIG
SNIPPERIG
WIPPERIG
MOPPERIG
POPPERIG
1SOPPERIG
2SOPPERIG
KNARPERIG
RASPERIG
NIJDASSERIG
JUDASSERIG
PIASSERIG
PISSERIG
PATSERIG
KLETSERIG

PROTSERIG
TERIG
GATERIG
SNATERIG
GRATERIG
WATERIG
1PIETERIG
2PIETERIG
PETIETERIG
KIEREWIETERIG
ROETERIG
SPROETERIG
TROETERIG
SALPETERIG
ZWETERIG
MUGGEZIFTERIG
SCHOFTERIG
HUFTERIG
JACHTERIG
SMACHTERIG
KLUITERIG
TUITERIG
SCHIJTERIG
MIJTERIG
PALTERIG
EELTERIG
FIELTERIG
SMELTERIG
KOMEDIANTERIG
KRANTERIG
ASTRANTERIG
DILETTANTERIG
SLENTERIG
PENTERIG
KRENTERIG
PRENTERIG
FLINTERIG
SPLINTERIG
KLONTERIG
STRONTERIG
IDIOTERIG
MALLOTERIG
STOTERIG
SMARTERIG
PLOERTERIG
KWASTERIG
ARTIESTERIG
NESTERIG
NOESTERIG
KNOESTERIG
ROESTERIG
HEISTERIG
PUISTERIG
GALSTERIG
BULSTERIG
DEEMSTERIG
GLINSTERIG
FATTERIG
KATTERIG

Figuur 1

LISP een atom. Om een idee te geven hoe dat gaat:

correct atom	incorrect atom
Van_Dale	Van Dale
eerstewoord	1ewoord
soms1ewoord	soms 1ewoord

LISP is een programmeertaal op een nogal hoog niveau, dat wil zeggen ze staat nogal dicht bij de gebruiker. Daardoor is het mogelijk dat een gehanteerd atom al een speciale betekenis heeft. Daarvoor moeten we oppassen, immers, die speciale betekenissen slaan altijd op taken.

Eén ervan is het atom SETQ, dit is de naam van een opdracht. Nu geldt in LISP altijd dat als een atom letterlijk genomen moet worden (dus ontdaan van enige speciale betekenis), er een aanhalingsteken geplaatst dient te worden vóór dat atom. Wanneer we willen dat de naam vandale letterlijk genomen moet worden, dan zetten we 'vandale. Hetzelfde geldt ook voor meer atoms tegelijk, teksten dus. Om een tekst onder de naam faust op te bergen in het computergeheugen typen we simpelweg in:

```
(SETQ faust    '(en hier komt de volledige faust))
```

Let erop dat deze tekst op zichzelf tussen haakjes staat en duizenden regels omvat. Voor de openingshaak staat een aanhalingsteken; voor uitleg zie tekst hierna!

Nu is voor de computer de tekst (die uit met haken bij elkaar gehouden losse atoms bestaat) letterlijk als tekst bekend. Let op het aanhalingsteken voor het tekstopeningshaakje. Stond dat er niet, dan zou LISP denken dat het atom 'en' voor een taak staat!

We spreken af dat de namen van standaardfuncties in LISP voortaan met hoofdletters geschreven zullen worden. Andere zaken schrijven we met kleine letters.

Terug naar het woordenboek van Van Dale. We leggen de inhoud van dit werk in het computergeheugen vast onder de naam vandale, waarbij we ons voorstellen dat ieder woord op een nieuwe regel komt:

```
(SETQ vandale      '(

a
aa
aag
aan
aak
aal
.....etc.
   ))
```

Let op dat er twee sluithaken moeten komen: één voor de woordenlijst en één voor het einde van de werking van de taak SETQ. Om te weten of er tegenover iedere openingshaak een sluithaak staat, heeft de LISP interpretator een boekhouder in dienst die de haakjes automatisch nummert. Als we die haakjesnummering activeren (door vooraf de P van parenthesis mee te delen), komt er voor (SETQ vandale '(aap noot mies)) de tweeregelige tekst:

```
(SETQ vandale   '(aap noot mies))
0               1              10
```

De haakjesnummering opent met nul en sluit er ook weer mee. In dit voorbeeld is er dus overeenkomst tussen het aantal openings- en sluithaken.

Als dat laatste niet het geval is komt er de melding:

UNMATCHED PARENTHESES

hetgeen betekent: 'niet passende haakjes'. Eigenlijk is dit passen van haakjes de enige grammaticale regel van de programmeertaal LISP. Het is dan ook de eenvoudigste programmeertaal die er is; daarom vindt men haar echter ook moeilijk: ze heeft nauwelijks beperkingen. Voorbeeld: het eerste woord uit een woordenboek kan een taak worden door het aanhalingsteken ervoor weg te laten. Zo eenvoudig is het om van data naar programma's over te schakelen. Eigenlijk is dat aanhalingsteken een taak die ook wel QUOTE genoemd wordt. Een formulering dus als (QUOTE (aap noot mies)) is een correcte LISP-uitdrukking, maar het vergt aanzienlijk meer werk dan het tikken van dat simpele aanhalingsteken! Vandaar dat we ons meestal beperken tot het laatste: we zullen overigens dat woord QUOTE ook maar gaan gebruiken, het is korter dan het woord aanhalingsteken.

We hebben inmiddels het woordenboek van Van Dale onder de naam vandale opgeslagen en gaan ons concentreren op een taak die we opblaas zullen noemen. Het woord taak vervangen we af en toe (voor de afwisseling) door woorden als operator, functie en opdracht. De taak opblaas is in LISP bekend onder de naam EXPLODE of EXPAND. Deze werkt zo (het resultaat van de werking staat rechts van de pijl):

```
(EXPLODE  'aap)          → (a a p)
(EXPLODE (QUOTE noot))   → (n o o t)
(EXPLODE  'mies)         → (m i e s)
```

We willen een retrograde woordenboek maken, daartoe moeten we alle woorden opblazen tot letters. Daarvoor is een functie nodig die EXPLODE herhaaldelijk toepast.

• De tweede stap op weg naar het retrograde woordenboek

Omdat de functie EXPLODE niet op meer woorden tegelijk kan werken (EXPLODE '(aap noot mies)) geeft een foutmelding van LISP, moeten we de woorden één voor één uit Van Dale halen om ze per stuk op te blazen. De taak om EXPLODE herhaald toe te passen wordt verricht door een functie MAPCAR; deze pakt automatisch elk eerstvolgende woord uit de Van Dale, blaast het woord op en legt alles tezamen vast door er haken omheen te zetten; de correcte instructie hiervoor is:

```
(MAPCAR  vandale  'EXPLODE)
```

Let op dat de naam vandale nu n i e t ge_quot_d wordt, immers, de w o o r d e n uit de Van Dale dienen opgeblazen te worden en niet de naam vandale zelf. De functie EXPLODE moet echter letterlijk toegepast worden. Het resultaat van bovenstaande opdracht is ((a a p) (n o o t)(m i e s)) indien Van Dale slechts '(aap noot mies) bevatte. We willen echter straks meer: we nemen ons in stap drie op weg naar ons einddoel voor, die functie te maken die alle woorden omkeert. Dit doen we straks door op het resultaat van de vorige MAPCAR, die de woorden opblies, een nieuwe MAPCAR los te laten die de woorden omkeert. Daarom leggen we nu het opgeblazen woordenboek vast onder de naam opblaasvandale:

```
(SETQ opblaasvandale (MAPCAR vandale  'EXPLODE))
```

Let erop dat er voor de openingshaak van MAPCAR geen QUOTE staat, anders zou het atom MAPCAR n i e t voor een functie staan.

In sommige LISP-systemen is de standaardfunctie MAPCAR niet aanwezig. Daarom geven we hem nu. Let op de operator CAR; deze pakt elke keer de eerste van de lijst y terwijl (CDR y) alles (inclusief de haakjes) behalve de CAR van y is.

```
(DEF (mapcar (y taak)(COND
   ((NULL y) NIL)
   (T (APPEND (taak (CAR y))
      (mapcar (CDR y) taak))))))
```

• De derde stap op weg naar de retrograde Van Dale

Op de opgeblazen Van Dale laten we opnieuw de functie MAPCAR los, om alle woorden om te keren. De hulpfunctie die we hiervoor gebruiken heet REVERSE en we leggen meteen het resultaat vast onder de naam omkeervandale:

```
(SETQ omkeervandale (MAPCAR opblaasvandale 'REVERSE))
```

Het resultaat staat nu onder de naam omkeervandale: ((p a a)(t o o n) (s e i m)), indien de Van Dale slechts '(aap noot mies) bevatte.

In enkele LISP-systemen wordt de functie REVERSE niet standaard geleverd. Daarom nemen we even de functiedefinitie van een reverse op:

```
(DEF (reverse (lijst)(COND
   ((NULL lijst) NIL)
   (T (APPEND (reverse (CDR lijst))
      (CONS (CAR lijst) NIL))))))
```

• De vierde stap op weg naar het retrograde woordenboek

Om de thans omgekeerde woorden te kunnen sorteren op alfabetische volgorde moeten we een sorteeralgoritme bedenken. Er zijn nogal wat wegen die we voor de oplossing van dit probleem kunnen gebruiken, omdat er veel manieren zijn om te sorteren (het alfabet van achter naar voren òf omgekeerd òf ergens in het midden van het alfabet beginnen òf sorteren op de meest voorkomende letters, etc.).

Voor ons probleem (het sorteren op alfabetische volgorde) nemen we de meest voor de hand liggende methode: van voren naar achter in het alfabet. We ontwerpen derhalve een algoritme voor een sorteermethode die een lijst van losse letters in alfabetische volgorde plaatst. Om dat te kunnen doen dienen we elke keer de eerstvolgende letter in het alfabet te vinden en die in een nieuwe lijst achteraan te plaatsen. Om dit proces voor te stellen en dan dus te kunnen zien wat er gedaan moet worden en in welke volgorde bestuderen we een praktisch voorbeeld en bekijken precies het patroon van de uitgevoerde operaties.

Neem de lijst (L, M, G, R, C). De achtereenvolgende sorteerhandelingen die we de computer willen laten verrichten om de alfabetische volgorde (C, G, L, M, R) te maken zijn in de tabel op p. 59 te vinden.

Laten we kolom (2) met de telkens overblijvende nog ruwe data benoemen met de naam ruw. Deze lijst ruw begint met de volledige data-lijst. Bij iedere sorteerronde is ruw een element kwijt. Laten we kolom (3) de naam voor geven, omdat de elementen daarin per sorteerronde telkens de voorste zijn (in alfabetische zin) uit de lijst ruw. Tenslotte benoemen we kolom (4) met de naam doel, omdat daarin het resultaat dient te komen waarnaar we bij dit sorteren streven. Doel is het resultaat van het hangen (in het Engels: appending) van het atom in voor achteraan de vorige inhoud van doel. Het beschreven proces van alfabetiseren kunnen we als volgt in LISP programmeren.

(1) sorteer-ronde	(2) lijst waarop gewerkt wordt	(3) eerstvolgende in alfabet	(4) lijst op alfabetische volgorde
	ruw	voor	doel
1	(L, M, G, R, C)	C	(C)
2	(L, M, G, R)	G	(C, G)
3	(L, M, R)	L	(C, G, L)
4	(M, R)	M	(C, G, L, M)
5	(R)	R	(C, G, L, M, R)
6	()		

Begin met de opmerking dat het een programma betreft en benoem de variabelen in een lijst; daarna komt de toewijzing van de gegevens (data) aan de werklijst ruw. Vervolgens komt de label 'alweer', waarna er een lus komt die achtereenvolgens vastlegt: de eerste van ruw wordt in voor geborgen waarna de lijst ruw ontdaan wordt (= EFFACE) van hetgeen zojuist in voor geladen is. De inhoud van doel wordt nu opnieuw bepaald door achter zijn reeds bestaande inhoud de inhoud van voor te hangen (= APPEND). Na de CONDitie dat de inhoud van de ruwe lijst niet leeg (= NULL) mag zijn (want dan precies moet de inhoud van doel weergegeven (= RETURN) worden en na een voltooide RETURN stopt het programma automatisch) volgt tenslotte de sprongopdracht: ga naar alweer. Het programma om te alfabetiseren luidt dan:

```
(PROG (ruw voor doel)
   (SETQ ruw data)
alweer
   (SETQ voor (eerste ruw))
   (SETQ ruw (EFFACE voor ruw))
   (SETQ doel (APPEND doel voor))
   (COND ((NULL ruw)(RETURN doel)))
   (GO alweer)))
```

Enkele vreemde dingen zitten nog in het bovenstaande algoritme alfabetiseer. De termen 'eerste', 'COND' en 'EFFACE' zijn onbekend. Met EFFACE zijn we snel klaar, het betekent 'veeg uit', dus (EFFACE voor ruw) betekent: veeg het atom dat thans in voor zit uit de lijst die nu in ruw zit. De term EFFACE heet in sommige computers DELETE, in andere ook wel DIFFLIST; enfin, de naam doet er niet zoveel toe.

De term COND is een standaardnaam voor de conditionalizer in

LISP. Dat is de als-dan-statement in veel andere talen. De als-dan-statement bestaat uit twee gedeelten, het als-gedeelte en het dan-gedeelte. Dat is in LISP ook zo:

```
(COND ((als aan deze voorwaarde voldaan is)
       (voer dan deze tweede uitdrukking tussen haakjes
        uit)))
```

Met de term "eerste" hebben we een probleem dat we zelf moeten oplossen, omdat we een operator nodig hebben die uitzoekt welke de eerste (in alfabetische zin) is in de lijst ruw.

De taak voor "eerste" uit de lijst ruw luidt heel precies: Als ruw leeg is neem dan NIL. Wanneer de rest van ruw leeg is, neem dan de voorste (= CAR). Neem anders de EERDERE (in alfabetische zin) van de voorste van ruw en de eerste uit de rest van ruw. (Als het getallen waren zou EERDERE de 'kleinere van twee' heten en eerste zou heten: het minimum van een rij getallen.)

In LISP luidt dit DEFiniëren van deze functie "eerste", die moet werken op ruw, als volgt:

```
(DEF (eerste (ruw)(COND
   ((NULL ruw) NIL)
   ((NULL (CDR ruw))(CAR ruw))
(T (eerdere (CAR ruw)(eerste (CDR ruw)))))))
```

Indien "alfabetiseer" aan het werk gezet wordt op lijsten van letters, zoals '(B, A, Q, W, E, R, T,) en '(F, A, G, G, I, N, G, Z, I, L, O, G, G,) komt er respectievelijk (A, B, E, Q, R, T, W,) en (A, F, G, G, G, G, G, I, L, N, O, Z,). De werkwijze is dus dat de eerstvolgende uit het alfabet telkens door middel van de functie "eerste" uit de ruwe data gepakt wordt en op de juiste plaats gezet. De functie "eerste" levert voor '(Z, W, E, R, K,) het resultaat E en voor '(Q, W, A, R, K,) de output A. Het proces om, alfabetisch gezien, de eerstvolgende uit de meegegeven lijst te vinden, kan geschieden door gebruik te maken van de functie "eerdere". Deze zorgt ervoor dat uit ieder paar letters zoals aan "eerdere" aangeboden, de eerdere gehaald wordt. (eerdere 'A 'Z) levert dus A, (eerdere 'Q 'B) levert B en (eerdere 'Z 'A) levert A. (eerdere 'S 'S) levert echter S.

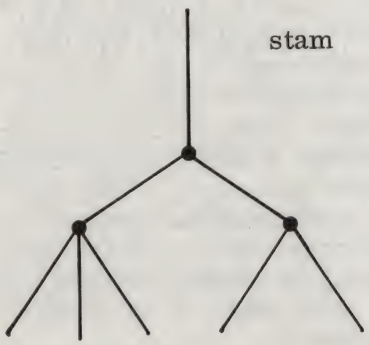
De functie eerdere moeten we nog definiëren. Dat kan eenvoudig met de functie MEMBER die standaard is in LISP-systemen. De MEMBER zorgt ervoor dat hij een element, bijvoorbeeld 'A, kan vinden in een lijst, bijvoorbeeld '(F, A, G, G, I, N). Zodra hij de eerste maal ziet dat het gevraagde atom voorkomt geeft hij als antwoord de lijst '(A, G, G, I, N). Komt het gevraagde atom niet voor, dan geeft hij NIL. De definitie van eerdere luidt nu:

```
(DEF (eerdere (a, b)(COND
   ((MEMBER a (MEMBER b alfabet)) b)
   (T a))))
```

De tweede uitdrukking in de conditional betekent in andere programmeertalen: if true then RETURN a, en omdat true altijd waar is zal de functie eerdere altijd de inhoud van het eerste meegegeven argument opleveren, indien de executie al niet gestopt is omdat in de vorige regel bleek dat de inhoud van het tweede argument eerder in het alfabet bleek te zitten.

Nu komt het moeilijkste gedeelte, namelijk het hanteren van het alfabetiseren op de eerste letters van de omgekeerde woorden van de Van Dale en vervolgens op de tweede letters etc. en tegelijkertijd het regelen van het sorteren van niet alleen de behandelde letters maar ook de rest van de woorden. En dat kan helemaal niet met het tot dusverre ontwikkelde alfabetiseer-algoritme. Immers dit algoritme kan de beginletters p, t en s van de omkeervandale ((p a a)(t o o n)(s e i m)) prima sorteren tot p, s en t op alfabetische volgorde. Maar hoe moet het dan met de rest van de woorden: ((a a)(o o n)(e i m)? Die zijn we gewoon kwijt!

Om dat probleem te voorkomen gaan we over op een methode van sorteren die wel 'boomsorteren' (= treesort) heet. De te sorteren woorden worden nu ondergebracht in een boom, die bestaat uit knooppunten (= nodes); ieder knooppunt wijst weer naar een ander knooppunt:



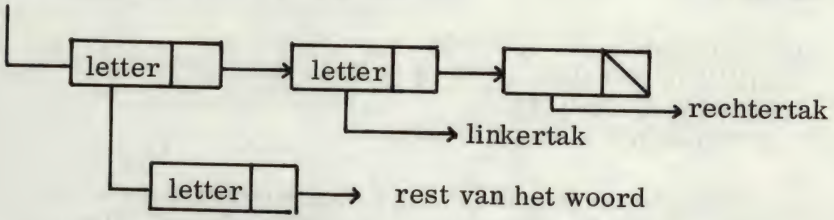
Beginnend met de stam van de boom (vergeleken met natuurlijke bomen staan sorteerbomen op hun kop!) is er een eenduidig pad naar ieder knooppunt. Bij de boomsorteermethode worden de woorden die gesorteerd moeten worden in de knooppunten van de boom geplaatst. Dit sorteren wordt zodanig gedaan dat bij het doorlopen van een gesorteerde boom in iedere linkertak van een knoop slechts woorden worden gevonden die (alfabetisch gezien) eerder komen dan die in die

knoop. Natuurlijk betekent dat dan, dat in de rechtervertakkingen van de knopen slechts woorden zitten die verderop in het alfabet komen dan die in de desbetreffende knopen.

Een LISP-programma voor dit sorteerwerk kan als volgt gemaakt worden. De te sorteren omgekeerde woorden uit de Van Dale zijn door de werking van MAPCAR tot lijsten gemaakt en er moet gesorteerd worden op de eerste letter, vervolgens op de tweede letter, etc.

((letter1, letter2, ...)(letter1, letter2, ...)(...) ...)

De eerste stap is nu het maken van een toevoegfunctie, die werkt met een woord x en een gedeeltelijk geconstrueerde sorteerboom z, waarvan het resultaat is dat aan die z op correcte wijze het woord x ingehangen is. De boom bestaat uiteraard uit knopen die ieder op hun beurt bestaan uit drie onderdelen:



Dus als w een knooppunt is dan is de eerste letter van het daarin opgeslagen woord te vinden door middel van (CAAR w), de linkertak is te vinden met (CADR w) en de rechtertak met (CADDR w).

De definitie van "voeg" wordt dan:

```
(DEF (voeg (x, z)(COND
   ((NULL z)(LIST x NIL NIL))
   ((eerder (CAR z)(CAAR z))
      (LIST (CAR z)(voeg x (CADR z))(CADDR z)))
   (T (LIST (CAR z)(CADR z)(voeg x (CADDR z)))))))
```

Vervolgens moet met alle woorden, zoals door MAPCAR met behulp van EXPLODE en daarna geREVERSEd, een boom opgebouwd worden. Dat doen we met de functie "bouw". Bouw voegt alle woorden van een woordenboek w toe aan de boom v. Die boom v mag best leeg (= NIL) zijn; dan begint bouw een boom te construeren voor de woordenlijst w. Dat doen we dan maar. We hebben de omgekeerde aap, noot en mies onder de naam 'omkeervandale' opgeslagen en proberen nu:

(bouw omkeervandale NIL)

Dan komt er:

```
VALUE:
   ((p, a, a)((s, e, i, m)((t, o, o, n) NIL NIL) NIL) NIL)
```

Dit kan alleen maar omdat we de functie "eerdere" vervangen hebben door de functie "eerder". De laatste is een 'predikaat', dat is een verzamelnaam voor functies die òf waar òf onwaar opleveren. In LISP: T of NIL. In primitieve LISP-systemen is bijvoorbeeld MEMBER een predikaat.

De waarheid van "element x is een lid van de lijst l" wordt geprogrammeerd met:

- Als l leeg is, geef dan NIL. Anders,
- Als x gelijk is aan de CAR van l, geef T. Anders,
- Neem de waarheid van "x is lid van de CDR van l".

In LISP luidt dit:

```
(DEF (member (x, l)(COND
   ((NULL x) NIL)
   ((EQ x (CAR l) T)
   (T (member x (CDR l)))))))
```

Dit was om te wennen aan predikaten. Nu programmeren we het predikaat "eerder". Dat hebben we nodig in de functie "voeg". Het moet zo werken dat er komt:

- a komt alfabetisch eerder dan b true
- b komt alfabetisch eerder dan a nil

```
(DEF (eerder (x, y)(COND
   ((MEMBER x (MEMBER y alfabet)) NIL)
   (T T))))
```

Indien een LISP-systeem toevallig de functie MEMBER ook als predikaat heeft, werkt bovenstaande definitie ook prima.

De vierde stap op weg naar de retrograde Van Dale is vrijwel voltooid indien we de functie "bouw" hebben:

```
(DEF (bouw (w, v)(COND
   ((NULL w) v)
   (T (bouw (CDR w) (voeg (CAR w) v))))))
```

zij het echter dat de resultaten van bouw een boomstructuur hebben. Daartoe 'vlakken' we de output van bouw af met de functie "vlakken":

```
(DEF (vlakken (a, b)(COND
   ((NULL a) b)
   (T (vlakken (CADR a)(CONS (CAR a)
      (vlakken (CADDR a) b)))))))
```

Hierin is a de af te vlakken sorteerboom, terwijl in b de gewone lijst van gesorteerde woorden opgeslagen worden. Als a leeg (= NULL) is, dus een knooppunt zonder vertakking, blijft b ongewijzigd. In de overige gevallen wordt het resultaat van het afvlakken van een linkervertakking toegevoegd aan de lijst die verkregen is door plakken (= CONS) van het woord op het onderhavige knooppunt aan hetgeen resulteert als de afgevlakte vorm van de rechtervertakking toegevoegd wordt aan de lijst b.

Het afvlakwerk start met b leeg:

```
(vlakken sorteerboom NIL)
```

Een sortering van de omgekeerde Van Dale op de eerste letter volgt nu met:

```
(vlakken (bouw omkeervandale NIL) NIL)
```

Dan komt:

```
VALUE:
   ((p, a, a)(s, e, i, m)(t, o, o, n))
```

Tenslotte moet nog gesorteerd worden op de tweede letter van de woorden. Breng daartoe de nodige veranderingen aan in dit programma en breid het programma zelf uit zodat het woorden van willekeurige lengte sorteert. Tip: de 'sleutel' waarop de functie voeg bij het sorteren let, zit in het bekijken van de eerste letter van een woord, breng daar dus verandering in.

Opgave 2 – Een woordenboek op maat (slaaf voor kruiswoordraadsels)

Een woordenboek op maat (= WOM) is een woordenboek dat woorden rangschikt in alfabetische volgorde, maar wèl zo dat de woorden die een gelijke lengte hebben eerst worden gealfabetiseerd. Een woordenboek op maat van de woorden:

- "theoretisch georiënteerde taalkundigen hebben klaarblijkelijk geen benul van computergebruik voor taalkundige toepassingen"

zou er als volgt uitzien:

lengte	woord
3	van
4	geen
	voor
5	benul
6	hebben
11	taalkundige
	theoretisch
12	toepassingen
	taalkundigen
13	georiënteerde
15	computergebruik
	klaarblijkelijk

Een dergelijk woordenboek zou wat handig zijn bij het oplossen van kruiswoordraadsels, die immers allemaal op woordlengte zijn opgebouwd.

Het programma WOM, zoals hier volgt, bekijkt slechts woorden van lengte 1 tot en met lengte 33 en gebruikt de standaardfunctie LENGTH om de lengte van geEXPLODE woorden te bepalen. Voor degenen die op hun LISP-systeem geen LENGTH hebben volgt hier een uitleg en een definitie.

De functie LENGTH is eenvoudig. We testen met de conditionalizer 'COND' of de tekst leeg (= NULL) is. Zo ja, dan heeft de tekst lengte 0; is de tekst niet leeg dan testen we of hij uit één atom bestaat. Zo ja, dan heeft de tekst de lengte 1. Tenslotte laten we voor alle andere gevallen (= T) één optellen bij de lengte van de rest (= CDR) van de tekst:

```
(DEF (LENGTH (tekst)(COND
   ((NULL tekst) 0)
   ((ATOM tekst) 1)
   (T (+1 (LENGTH (CDR tekst)))))))
```

Vragen we nu aan LISP om te evalueren:

```
(SETQ tekst '(a, a, p, noot, m, i, e, s))
(LENGTH tekst)
```

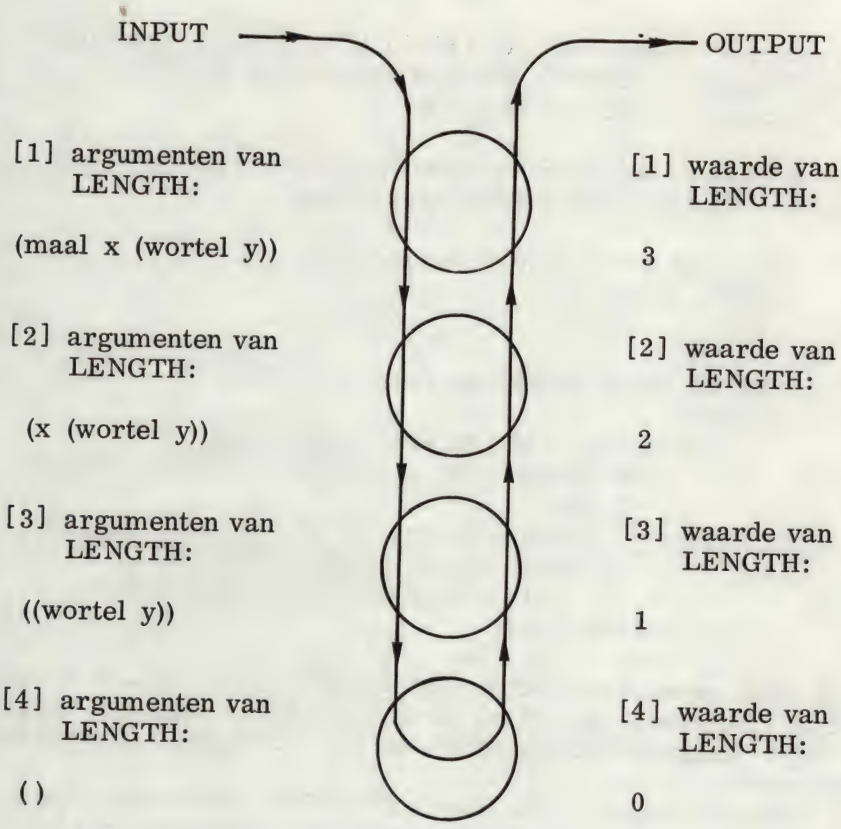
dan komt er 10. En zo hoort het ook.

Interessant wordt het indien we tekst de volgende inhoud geven:

```
(SETQ tekst '(maal x (wortel y)))
```

De LISP machine geeft na (LENGTH tekst) nu het antwoord: 3.

Zo hoort het niet! Wat is er gebeurd? Om daar achter te komen gaan we natrekken (= TRACE) wat er gebeurde tijdens de executie van (LENGTH tekst) door LISP. Daaruit destilleren we de volgende gebeurtenissen die gelezen moeten worden in de volgorde van de pijlen in figuur 2.



Figuur 2. Een trace van de recursieve functie LENGTH, toen deze geëxecuteerd werd met het argument (maal x (wortel y)).

Uit de trace van figuur 2 op een tekst, waarin onderdelen van de tekst tussen haken staan, blijkt dat vanwege de werking van de CDR in de definitie van LENGTH juist voor deze omhaakte subteksten als lengte = 1 genomen wordt. Om dat euvel op te heffen bouwen we een CAR in die ook recursief gehanteerd wordt. Het euvel is aldus verholpen:

```
(DEF (LENGTH2 (tekst)(COND
   ((NULL tekst) 0)
   ((ATOM tekst) 1)
   (T (+ (LENGTH2 (CAR tekst))
         (LENGTH2 (CDR tekst))
)))))
```

De trace van deze nieuwe definitie LENGTH2 op TEXT2 is wat ingewikkelder. Deze is te vinden in het boek van Patrick Winston: Artificial Intelligence (zie literatuurlijst).

De functie die een woordenboek op maat uit een ruwe geEXPLODE woordenlijst produceert, kan er als volgt uitzien:

```
(DEF (wom (ruw) (PROG (lengte x)
   (SETQ lengte 1)
   (SETQ x ruw)
lus
   (SETQ lengte (ADD1 lengte))
   (COND
      ((EQN lengte 34) (RETURN '(dit is het wom)))
      (  (NOT (MAPC ruw (FQUOTE (LAMBDA (woord)
            (COND
               ((NULL woord) NIL)
               ((EQN lengte (LENGTH woord))
                  (PRINT (COMPRESS woord))))))))
         (GO lus))))))
```

De functie EQN is een standaardfunctie die getallen met elkaar vergelijkt, terwijl EQ atoms met elkaar vergelijkt. EQ en EQN zijn in de meeste LISP-systemen van het predikaat-type. De functie ADD1 telt 1 op bij lengte.

Een beetje verrassend is de functie FQUOTE. Dat is hier handig omdat de MAPC herhaaldelijk (een soort MAPCAR dus) op "ruw" de operaties wil loslaten:

- kijk of er een woord is; zo nee, geef dan NIL; zo ja:
- vergelijk de te sorteren lengte met de LENGTH van het onderhanden woord en druk, indien deze lengtes hetzelfde zijn, het weer samengeperste (= COMPRESS) woord af.

De functie COMPRESS heet in sommige systemen MAKENAME of afgekort MKNAM en doet het tegenovergestelde van EXPLODE. In alle LISP-systemen geeft de PRINT-opdracht een afdruk aan het begin van een regel en daarna wordt er onmiddellijk een nieuwe regel genomen voor een eventuele volgende PRINT-opdracht. Zo komen de woorden netjes onder elkaar te staan. Dit zou niet kunnen met de

gewone MAPCAR, want die hangt (= APPEND) alle resultaten van zijn werking achter elkaar.

Daarom zit MAPC een beetje anders in elkaar. Hij wordt geconstrueerd met de PROGram operator; overigens is MAPC standaard in de meeste LISP-systemen aanwezig:

```
(DEF (mapc (woorden taak)(PROG( )
   lus    (COND ((NULL woorden)(RETURN NIL)))
          (taak (CAR woorden))
          (SETQ woorden (CDR woorden))
          (GO lus))))
```

Een aardigheidje in deze definitie zit in de CONDitie dat als er geen woorden meer zijn om te behandelen, er dan NIL gegeven wordt. Dit maakt het voor het programma wom gemakkelijk: het programma voor het woordenboek op maat reageert precies op deze NIL, om dan weer in de lus te gaan en dus de lengte van de te selecteren woorden met 1 op te hogen, totdat de lengte 34 bereikt is.

Opgave 3 – Kortere schrijfwijze

Hoe zou de werking van de lengtefunctie length (in zijn uitgebreide versie zoals hiervoor gegeven in opgave 2) bewerkstelligd kunnen worden in LISP met behulp van de eenvoudigste LENGTH functie en de MAPCAR?

Bestudeer de countnumbers zoals gegeven door Siklóssy in zijn boek: "Let's Talk Lisp", zie literatuurlijst.
Vereenvoudig deze in belangrijke mate.

Opgave 4 – De 'diepte' van een grammaticale structuur

We kunnen eenvoudig de diepte van een grammaticale structuur berekenen. Met 'diepte' bedoelen we het maximale aantal haakjes dat een woord in een tekst omgeeft. Daartoe gebruiken we uiteraard de standaardoperator MAX:

```
(MAX -1 2 3.16 5 -4)
```

heeft de waarde

```
5.
```

De bijbehorende functie luidt:

```
(DEF (diepte (zin)(COND
   ((NULL zin) 1)
   ((ATOM zin) 0)
   (T (+ 1 (MAX
      xxxxx
```

Nu kunnen we de functie EXPLODE gebruiken om de woorden in een tekst op te blazen en daarna kan het langste woord in een tekst gevonden worden:

```
(SETQ text '(er was)))

(SETQ opblaas (MAPCAR text 'explode))

(SETQ woordl (MAPCAR opblaas 'lengte))
```

Om de maximum woordlengte eruit te krijgen moeten we "woordl" bestrijken met een MAPCAR die twee argumenten tegelijk aankan. In sommige LISP-systemen is dat de EVALQUOTE-functie. In andere de APPLY-functie. We kiezen de eerste:

```
(EVALQUOTE woordl 'MAX)
```

2.5 ELIZA of "HOW IS YOUR SEX LIFE": BOEKHOUDEN EN CONVERSEREN

Automatische text processing wordt omgeven met een serie anekdotes. Een ervan gaat als volgt. Er was eens een computerbedrijf met een vice president. Deze vice president was zeer ijverig. Hij werkte nogal eens buiten de tijden dat andere employees aanwezig waren. Zijn contacten met collega's en andere menselijke wezens verliepen dan via elektronische weg. Op een avond ontspon zich zo het volgende gesprek achter de computer terminal (VP = vice president; TT = antwoord dat op de terminal werd gegeven; TelComp = Tele-Computing, dat is boekhouden met een computer die ver weg staat en met de terminal via een telefoonlijn bediend wordt):

VP: Denk je dat ik vanavond de computer kan gebruiken?
TT: Waarom vraag je dat?
VP: Dan zou ik wat extra kunnen werken aan de demonstratie van TelComp.
TT: Waarom wil je dat?
VP: Mijn toekomstige klanten hebben nog nooit een demonstratie van Tel Comp gezien.
TT: Is het belangrijk voor je?
VP: Ja natuurlijk.
TT: Daar lijk je nogal zeker van.
VP: Wat een belachelijke opmerking.
TT: Wat bedoel je met "belachelijke opmerking"?
VP: Kun je soms een reden aangeven waarom ik de computer niet kan gebruiken?
TT: Vertel me eens wat meer over je toekomstige klanten.
VP: Bel me maar op op nummer 491 - 1850 als je zover bent.

De VP loopt nu weg van het beeldscherm en belt woedend een van zijn nietsvermoedende collega's op.
Gesprek VP en Collega:

VP: Waarom doe je zo lullig tegen me?
Coll.: Hoezo? Wat bedoel je met "lullig tegen je zijn"?

De verklaring van het geheel was de volgende.

In de computer van de firma waar de vice president werkte was net een nieuwe vinding ingeplant, namelijk een computerprogramma waarmee men kon converseren. De conversatie ging op de manier van een Rogeriaanse therapeut, dat wil zeggen gedurende het gesprek gaat de therapeut zo op de uitlatingen van de cliënt in dat iedere opmerking van de cliënt wordt teruggekaatst. Voor veel mensen geeft dat een gevoel van warmte, omdat ze zich in hun eigen woorden zo goed begrepen voelen. De therapeut blijft buiten schot. Een dergelijk gesprek wordt kennelijk als behoorlijk echt ervaren, gezien de reacties van de VP in de anekdote.

Het programma waar het hier om gaat heet ELIZA en werd geschreven door Weizenbaum. In welke mate het programma een overtuigende indruk weet te maken, kan men bestuderen uit onderstaande dialoog die onlangs met een in Amsterdam aanwezige ELIZA werd gevoerd.

Tell me about your problem. Speak in complete (but simple) sentences.
Good afternoon.
Let's change the subject.
Good night.
Go on.
Bad night.
Is there anything else?
Yes.
Really?
Indeed.
I wish I were as sure of myself as you seem to be.
Interesting.
What are you most interested in life?
You.
Please explain.
It is simply like that.
What makes you happy?
You.
I'm still listening.
That's good.
Tell me something else.
Why?
Please continue.
Why?
Tell me about your aches?
Which of them?
Tell me about your emotions?
Which of them?
Are you afraid of being alone?
No.
Why not?
It is simply like that.
Tell me about your hobbies.
Which of them?
Would your mother have allowed that?
What?
Why do you ask that question?
Because of your previous question.
Is that the real reason?
Yes.
Tell me more.
Of what?
What is it you are getting at?
The same as you do.
Tell me more.
Of what?
What answer would you please the most?
Your answer.
Where does it hurt?
It does not.
Do you think people should be allowed to do whatever they want?
Yes.

I'm not sure I understand.
Neither do I.
Don't other reasons come to mind?
No.
Have you tried?
Yes.
I don't believe you.
Why?
What worries you?
Nothing.
Nothing?
Indeed.
That's interesting.
Why?
Honestly?
Answer my question.
Have you tried meditation?
Why not?
What else do you have to add?
Nothing.
Let's talk about your family?
Which of them?
Why don't you just pay your bill and go home?
Because you have to answer my question first.
Does that explain things?
Yes.
What makes you think so?
Your way of avoiding answers.
What are your feelings about loneliness?
Nothing.
Continue, please.
Please answer my first question.
What other reasons might there be?
You are repeating the same things.
Maybe I am repeating the same things.
Why are you doing so?
Does your mother feel that way?
I'm not able to judge that.
Why are you not able to judge that?
Because she is not here.
I am confused.
Why?
What are you afraid of?
You are repeating the same question without answering my previous one.
Perhaps I only pretend I am repeating the same question without answering your one.
Possible.
Please be more definite.
Definitly.
Go on.
With what?

What are your feelings right now?
Nothing. I am thinking.
What was your father like?
You are avoiding a real dialog.
How do you know I am avoiding a real dialog?
Because you make remarks not belonging to the discussion.
What are your true feelings?
About what?
Could you be more precise?
No.
How come?
About what?
Tell me about your sex life?
Which of them?
Just continue talking?
Why do you use a question mark at the end of your command?
How would you describe your personality?
I would not.
I don't understand.
Let us stop the discussion.
Would your father agree?
Of course.
Is your life satisfactory?
Of course.
Do you expect me to believe that?
Of course.
How is your health?
And yours?
Explain.
You have to explain now.
Why now?
Because you should answer my question.
Go on talking, its your money.
How can we stop?
Surely you know hoe.
Why do you use a dutch word?
You are a terrible insecure and indecisive person.
Are you sure?
How do you feel about death?
Good.
You seem more positive now.
Indeed.
I see.
So do I.
Please be honest with me.
Thats what I am.
I think you are sick, leave me alone.
I would be pleased to do so.
I understand.
So do I.
If you could change something in your life, what would it be?
This conversation.
You don't make any sense of all. Try fortran.

- Waarom ELIZA bij 'boekhouden' hoort

Om inzicht in de werking van het programma ELIZA te krijgen volgt hier eerst een serie vragen en opgaven.

1. De belangrijkste opgave is: maak uit, welke uitspraken door de computer worden gedaan en welke door degene die achter de terminal zat.
2. Maak een overzicht van de antwoorden die zowel van de computer als van de terminal-bediener zouden kunnen zijn.
3. Waar stokt de conversatie?
4. Wat gebeurt er wanneer er kennelijk geen antwoord is voor de computer?
5. Merk op de spelfouten tegen het Engels. Wat is daar voor bijzonders mee?
6. Bedenk een eenvoudig LISP-programma dat ELIZA kan simuleren. Als het te moeilijk blijkt, bestudeer dan de desbetreffende passages in Winston: "Artificial Intelligence" (zie literatuurlijst).

Een overzicht van alle antwoorden op de vragen hierboven zou het volgende beeld moeten opleveren. De situatie in de dialoog is zo dat het niet altijd even duidelijk is wie er nu precies aan het woord is, de computer of de proefpersoon. Dat komt doordat de computer meer dan eens òf precies reageert op een woord uit de zin van de proefpersoon, òf hij geeft een totaal ander antwoord, zo in de trant van: wat zit je nu toch moeilijk te doen! Verder reageert de computer als hij niets te zeggen weet met een aanmoediging van 'ga rustig door, ik luister'.

Moeilijk wordt het voor de computer wanneer hij een v r a a g voorgelegd krijgt. Ook dat is duidelijk te volgen in het verslag van de dialoog.

Impliciet zit in het bovenstaande het antwoord op de vraag, waarom ELIZA bij boekhouden hoort. ELIZA is een boekhoudprogramma omdat het alleen maar op een standaardmanier reageert op vaste woorden in een uiting van zijn dialoogpartner. Dat dat een sterk op de werkelijkheid van menselijke conversatie gelijkende indruk maakt, ligt niet aan de computer, maar veeleer aan het feit, dat kennelijk mensen elkaar op deze mechanische wijze benaderen. Het schijnt zelfs zo te zijn dat deze manier van 'aandacht voor elkaar' wordt beloond, immers de cliënt die alsmaar zijn eigen woorden uit de mond van zijn therapeut verneemt, begint zich veilig en begrepen te voelen in veel situaties.

- Zelf ELIZA programmeren

Om zelf een ELIZA te programmeren hebben we meer programmeerkennis nodig dan we tot nu toe in dit boek hebben aangebracht. ELIZA is in feite een patroonherkennend programma. Patroonherkennende programma's schrijven we verder in dit boek in de patroonherkennende taal METEOR die in hoofdstuk 3 aan de orde komt. Daar zullen we dan ook op ELIZA terugkomen.

We sluiten deze paragraaf met een ELIZA-programma in BASIC. Daarna zullen we ons als afsluiting van dit hoofdstuk bezig houden met een andere bron van anekdotes, namelijk automatisch vertalen volgens de boekhoudmethode.

```
13REM-----------------------------------------------------
 20 REM -ELIZA- IN EEN BEWERKING VAN:
 40 REM -GUUS WISELIUS-
59Q1=0
60 PRINT"TEST SWITCH AAN? (J/N)"
61 INPUTI$
62 IFI$="J"THENQ1=1
 63  PRINT
 64  PRINT
 80 REM
 100 PRINT "     *** ELIZA ***"
 101  PRINT
 102  PRINT
 103  PRINT
 104  PRINT
 105  PRINT
 120 PRINT "MIJN DENKTIJD IS TOT +/- 30 SECONDEN **"
 121  PRINT
 140 Q$ = ".*?!"
 141  PRINT Q$
 160 PRINT
 161  PRINT
 162  PRINT
 163  PRINT
 164  PRINT
 165  PRINT
 166  PRINT
 180 DIM S(36),R(36),N(36)
 200 RESTORE
 220 N1 = 36
 221 N2 = 12
 222 N3 = 112
 240 FOR X = 1 TO N1 + N2 + N3
 241  READ Z$
 242  NEXT X
 260 FOR X = 1 TO N1
 280 READ S(X),L
 281 R(X) = S(X)
 282 N(X) = S(X) + L - 1
 300 NEXT X
 320 READ I$
 340 PRINT I$
 360 REM
 380 REM
 400 PRINT
 420 PRINT CHR$ (30);
440 INPUT I$
441 I$=">  "+I$+"  <<"
 460 PRINT
 500 FOR L = 1 TO LEN (I$)
 520 IF SUBSTR (I$,L,1) = "'" THEN I$ = SUBSTR (I$,1,L-1) + SUBSTR (I$,L
+1)
 540 IF L + 4 < = LEN (I$) THEN 542
 541 GOTO 560
 542 IF SUBSTR (I$,L,4) = "SHUT" THEN 544
 543 GOTO 560
 544 PRINT "SHUT UP..."
545 GOTO4420
 560 NEXT L
```

```
 580 IF I$ = P$ THEN 582
 581 GOTO 600
 582 PRINT "PLEASE DON'T REPEAT YOURSELF!"
 583  GOTO 360
 600 PRINT "    ";I$
 640 RESTORE
 661 F$ = ""
 662 S = 0
 700 FOR K = 1 TO N1
 710 PRINT ".";
 720 READ K$
 740 L8 = LEN (K$)
 780 L9 = LEN (I$) - LEN (K$) + 1
 800 FOR L = 1 TO L9
 820 IF SUBSTR (I$,L,L8) = K$ THEN 822
 821 GOTO 840
 822 F$ = K$
 823 S = K
 824 T = L
 840 NEXT L
 860 IF S > 0 THEN 862
 861 GOTO 880
 862 K = S
 863 L = T
 864  GOTO 960
 880 REM
 900 NEXT K
 920 IF S = 0 THEN 922
 921 GOTO 940
 922 K = 36
 923  GOTO 1580
 940 K = S
 941 L = T
 959 REM F$=HET GEVONDEN KEYWOORD
 960 REM L=PLAATS WAAR'T GEVONDEN KEYWOORD BEGINT
 961 REM K=HET BETREFFENDE KEYWOORD (DE PLAATS DAARVAN IN DE KEYWOORDENTA
BEL
1000 RESTORE
1010 IFQ1=1THENPRINT 1010;F$;L;K
1020 FOR X = 1 TO N1
1021  READ Z$
1022  NEXT X
1040 IF F$ = "" THEN 1120
1100 C$= " "+ SUBSTR (I$,L+LEN(F$))
1120IFQ1=1THENPRINT1120;L9;C$
1150 REM ******* CONJUGATE WOORDEN ZOEKEN **********
1180 FOR X = 1 TO N2 / 2
1200READS$
1201READR$
1220 S9 = LEN (S$)
1240 R9 = LEN (R$)
1260 B = 1
1270
1280 FOR L = B TO LEN (C$)
1300 C9 = LEN (C$)
1320 IF L + S9 > C9 THEN 1420
1340 IF SUBSTR (C$,L,S9) < > S$ THEN 1420
1360 C$ = SUBSTR (C$,1,L-1) + R$ + SUBSTR (C$,LEN(C$)-C9+L+S9+1)
1365 PRINT"C";
1370 IFQ1=1THENPRINT 1370;X;C$;R$;S$
1380 B = L + R9
1400 GOTO 1280
1420 IF L + R9 > C9 THEN 1500
1440 IF SUBSTR (C$,L,R9) < > R$ THEN 1500
1460 C$ = SUBSTR (C$,1,L+1) + S$ + SUBSTR (C$,LEN(C$)+C9+L+R9+1)
1480 B = L + S9
1481  GOTO 1280
1500 NEXT L
1520 NEXT X
1540 IF SUBSTR (C$,1,1)<>" "THEN1580
1560 C$=SUBSTR(C$,2)
1561GOTO1540
1580 REM
1600 REM
1620 RESTORE
1621  FOR X = 1 TO N1 + N2
1622  READ Z$
1623  NEXT X
1640 FOR X = 1 TO R(K)
1641  READ A$
1660 NEXT X
1680 REM
```

```
1700IFQ1=1THENPRINT 1700;S(K);R(K);N(K);K
 1720 PRINT
1721 PRINT
1722 PRINT ">> ";
 1740 R(K) = R(K) + 1
 1741  IF R(K) > N(K) THENR(K) = S(K)
1750IFQ1=1THENPRINT 1750;A$;1750;LEN(A$)
1755
 1760 IF SUBSTR (A$,LEN(A$)) < > "*" THEN 1762
 1761 GOTO 1780
 1762 PRINT A$
 1763  GOTO 1800
 1780 PRINT SUBSTR (A$,1, LEN (A$) - 1);" ";C$
 1800 P$ = 1$
 1801  GOTO 400
 1820 REM
 1840 REM
 1860 DATA CAN YOU,CAN I,YOU ARE,YOURE,I DONT,I FEEL
 1880 DATA WHY DONT YOU,WHY CANT I,ARE YOU,I CANT,I AM
 1900 DATA "IM ","YOU ",I WANT,WHAT,HOW,WHO,WHERE,WHEN,WHY
 1920 DATA NAME,CAUSE,SORRY,SEX,HELLO,HALLO,MAYBE
 1940 DATA "NO ",YOUR,ALWAYS,THINK,ALIKE,YES,FRIEND
 1960 DATA "COMPUTER",NOKEYFOUND
 1980 REM
 2000 DATA " ARE ", AM ,WERE ,WAS , YOU , I ,YOUR ,"MY "
 2020 DATA " IVE ", YOUVE , IM ," YOURE "
 2040 REM
 2060 DATA DON'T YOU BELIEVE THAT I CAN*
 2080 DATA PERHAPS YOU WOULD LIKE TO BE ABLE TO*
 2100 DATA YOU WANT ME TO BE ABLE TO*
 2120 DATA PERHAPS YOU DON'T WANT TO*
 2140 DATA DO YOU WANT TO BE ABLE TO*
 2160 DATA WHAT MAKES YOU THINK I  AM*
 2180 DATA DOES IT PLEASE YOU TO BELIEVE I AM*
 2200 DATA PREHAPS YOU WOULD LIKE TO BE*
 2220 DATA DO YOU SOMETIMES WISH YOU WERE*
 2240 DATA DON'T YOU REALLY*
 2260 DATA WHY DON'T YOU*
 2280 DATA DO YOU WISH TO BE ABLE TO*
 2300 DATA DOES THAT TROUBLE YOU?
 2320 DATA TELL ME MORE ABOUT SUCH FEELINGS
 2340 DATA DO YOU OFTEN FEEL*
 2360 DATA DO YOU ENJOY FEELIN*
 2380 DATA DO YOU REALLY BELIEVE I DON'T*
 2400 DATA PERHAPS IN GOOD TIME I WILL*
 2420 DATA DO YOU WANT ME TO*
 2440 DATA DO YOU THINK YOU SHOULD BE ABLE TO*
 2460 DATA WHY CAN'T YOU*
 2480 DATA WHY ARE YOU INTERESTED IN WHETHER OR NOT I AM *
 2500 DATA WOULD YOU PREFER IF I WERE NOT*
 2520 DATA PERHAPS IN YOUR FANTASIES I AM*
 2540 DATA HOW DO YOU KNOW YOU CAN'T*
 2560 DATA HAVE YOU TRIED?
 2580 DATA PERHAPS YOU CAN KNOW*
 2600 DATA DID YOU COME TO ME BECAUSE YOU ARE*
 2620 DATA HOW LONG HAVE YOU BEEN*
 2640 DATA DO YOU BELIEVE IT IS NORMAL TO BE*
 2660 DATA DO YOU ENJOY BEING*
 2680 DATA WE WERE DISCUSSING YOU--NOT ME.
 2700 DATA OH - DO I*
 2720 DATA YOURE NOT REALLY TALKING ABOUT ME - ARE YOU?
 2740 DATA WHAT WOULD IT MEAN TO YOU IF YOU GOT*
 2760 DATA WHY DO YOU WANT*
 2780 DATA SUPPOSE YOU SOON GOT*
 2800 DATA WHAT IF YOU NEVER GOT*
 2820 DATA I SOMETIMES ALSO WANT*
 2840 DATA WHY DO YOU ASK?
 2860 DATA DOES THAT QUESTION INTEREST YOU?
 2880 DATA WHAT ANSWER WOULD YOU PLEASE THE MOST?
 2900 DATA WHAT DO YOU THINK?
 2920 DATA ARE SUCH QUESTIONS IN YOUR MIND OFTEN?
 2940 DATA WHAT IS IT WHAT YOU REALLY WANT TO KNOW?
 2960 DATA HAVE YOU ASKED ANYONE ELSE?
 2980 DATA HAVE YOU ASKED SUCH QUESTIONS BEFORE?
 3000 DATA WHAT ELSE COMES IN YOUR MIND WHEN YOU ASK THAT?
 3020 DATA NAMES DON'T INTEREST ME.
 3040 DATA I DON'T CARE ABOUT NAMES--PLEASE GO ON.
 3060 DATA IS THAT THE REAL REASON?
 3080 DATA DON'T ANY OTHER REASONS COME TO MIND?
 3100 DATA DOES THAT REASON EXPLAIN ANYTHING ELSE?
 3120 DATA WHAT OTHER REASONS MIGHT THERE BE?
 3140 DATA PLEASE DON'T APOLOGIZE!
 3160 DATA APOLOGIES ARE NOT NECESSARY.
 3180 DATA WHAT FEELINGS DO YOU HAVE WHEN YOU APOLOGIZE?
```

```
3200 DATA DON'T BE SO DEFENSIVE!
3220 DATA DO YOU LIKE FUCKING?
3240 DATA ARE YOU A HOMOSEXUAL?
3260 DATA WHAT PERSONS WOULD YOU LIKE TO FUCK?
3280 DATA WHAT DO YOU FEEL WHILE MAKING LOVE?
3300 DATA HOW DO YOU DO... PLEASE STATE YOUR PROBLEM.
3320 DATA YOU DON'T SEEM QUITE CERTAIN.
3340 DATA WHY THE UNCERTAIN TONE?
3360 DATA CAN'T YOU BE MORE POSITIVE?
3380 DATA YOU AREN'T SURE?
3400 DATA DON'T YOU KNOW?
3420 DATA ARE YOU SAYING NO JUST TO BE NEGATIVE?
3440 DATA YOU ARE BEING A BIT NEGATIVE.
3460 DATA WHY NOT?
3480 DATA ARE YOU SURE?
3500 DATA WHY NO?
3520 DATA WHY ARE YOU CONCERNED ABOUT MY*
3540 DATA WHAT ABOUT YOUR OWN *
3560 DATA CAN YOU THINK OF A SPECIFIC EXAMPLE?
3580 DATA WHEN
3600 DATA WHAT ARE YOU THINKING OF?
3620 DATA "REALLY, ALWAYS?"
3640 DATA DO YOU REALLY THINK SO?
3660 DATA BUT YOU ARE NOT SHURE YOU*
3680 DATA DO YOU DOUBT YOU*
3700 DATA IN WHAT WAY?
3720 DATA WHAT RESEMBLANCE DO YOU SEE?
3740 DATA WHAT DOES THE SIMILARITY SUGGEST TO YOU?
3760 DATA WHAT OTHER CONNECTIONS DO YOU SEE?
3780 DATA COULD THERE REALLY BE SOME CONNECTION?
3800 DATA HOW?
3820 DATA YOU SEEM QUITE POSITIVE.
3840 DATA ARE YOU SHURE?
3860 DATA AH.. I SEE
3880 DATA I UNDERSTAND
3900 DATA WHY DO YOU BRING UP THE TOPIC OF FRIENDS?
3920 DATA DO FRIENDS WORRY YOU?
3940 DATA DO FRIENDS PICK ON YOU?
3960 DATA ARE YOU SURE YOU HAVE ANY FRIENDS?
3980 DATA DO YOU IMPOSE ON YOUR FRIENDS?
4000 DATA PERHAPS YOUR LOVE FOR FRIENDS WORRIES YOU.
4020 DATA DO COMPUTERS WORRY YOU?
4040 DATA ARE YOU TALKING ABOUT ME IN PARTICULAR?
4060 DATA ARE YOU FRIGHTENED BY MACHINES?
4080 DATA WHY DO YOU MENTION COMPUTERS?
4100 DATA WHAT HAVE MACHINES TO DO WITH YOUR PROBLEM?
4120 DATA DON'T YOU THINK COMPUTERS CAN HELP PEOPLE?
4140 DATA WHAT IS IT ABOUT MACHINES THAT WORRIES YOU?
4160 DATA SAY-DO YOU HAVE ANY PSYCHOLOGICAL PROBLEMS?
4180 DATA WHAT DOES THAT SUGGEST TO YOU?
4200 DATA PLEASE CONTINUE
4220 DATA I'M NOT SURE THAT I UNDERSTAND YOU FULLY.
4240 DATA COME COME ELUCIDATE YOUR THOUGHTS.
4260 DATA CAN YOU ELABORATE ON THAT?
4280 DATA THAT IS QUITE INTERESTING.
4300 REM
4320 DATA 1,3,4,2,6,4,6,4,10,4,14,3,17,3,20,2,22,3,25,3
4340 DATA 28,4,28,4,32,3,35,5,40,9,40,9,40,9,40,9,40,9,40,9
4360 DATA 49,2,51,4,55,4,59,4,63,1,63,1,64,5,69,5,74,2,76,4
4380 DATA 80,3,83,7,90,3,93,6,99,7,106,7
4400 DATA HI ! I'M ELIZA.WHAT IS  YOUR PROBLEM?..
4420 END
```

2.6 HET VLEES IS WEL GEWILLIG MAAR DE GEEST IS ZWAK: OVER AUTOMATISCH VERTALEN WOORD VOOR WOORD

Evenmin als de ELIZA-anekdote mag de vertaalmop ontbreken in een boek over text processing. Er zijn er verschillende, allemaal verzonnen uiteraard, zij het dat ze wel voor waar en echt gebeurd doorgaan.

Zo gaat de mare dat er een vertaalmachine was ontwikkeld voor de vertaling van het Engels naar het Russisch en andersom. Aan deze machine werd toen hij voor volwaardig doorging de beroemde uitspraak uit Marcus 14:38 voorgelegd: "Spiritus quidem promptus est caro vero infirma." In het Nederlands: "De geest is wel gewillig, maar het vlees is zwak." De opgave was: vertaal de zin eerst vanuit het Engels in het Russisch en dan weer terug vanuit het Russisch in het Engels. De uitslag helemaal aan het eind was: "De whisky is behoorlijk, maar de biefstuk is slap."

Deze anekdote wordt steevast gebruikt om aan te tonen dat het nooit zal lukken automatisch vertalingen te maken. De redenering is dan meestal dat voor een automatische vertaling de kennis van alle tijden, heden, verleden en toekomst nodig is en dat die kennis nooit in een computer kan worden opgeslagen, vooral de kennis van de toekomst niet natuurlijk.

Ook wordt deze anekdote gebruikt om de niet ter zake kundige leek voor te rekenen dat zinnen die men moet vertalen een paar honderd betekenissen hebben. En inderdaad, wanneer men alle mogelijke betekenissen van alle afzonderlijke woordenboeken via een boekhoudsysteem noteert en met elkaar vermenigvuldigt, komt men gemakkelijk tot zulke dwaze getallen. Dwaas, omdat geen mens zich bij het vertalen ooit alle betekenissen van alle woorden voor de geest haalt.

Er is kennelijk wel wat anders aan de hand wanneer we vertalen. Vertalen is kennelijk een proces van voortdurend beslissingen nemen en niet een proces van voortdurend vermenigvuldigingen maken. Het is trouwens in de whisky-anekdote al direct heel duidelijk dat de 'behoorlijke whisky' niet op koshere wijze tot stand kan zijn gekomen. Immers de combinatie van sterke drank en gewillig past niet op elkaar. Dat wil dus zeggen dat achter de anekdote van de gewillige geest een primitief model van vertalen zit. Degenen die met instemming deze anekdote navertellen, verenigen zich kennelijk met het idee dat vertalen een kwestie is van woordenboeken vergelijken en verder zo ongeveer niets. Dan wordt vertalen woord voor woord aan elkaar plakken. Dan wordt vertalen boekhouden.

Een vertaalmachine die niets anders doet dan woorden uit het ene woordenboek vervangen door woorden uit het andere woordenboek kunnen we als volgt met de middelen die we tot nu toe in LISP hebben programmeren.

De input moet zijn het Latijns-Nederlands woordenboek met de volgende ingangen (ENTRIES): spiritus = geest; quidem = wel;

promptus = vastberaden, bereidwillig; est = is; caro = vlees; vero = evenwel; infirma = zwak, slap. De organisatie van het programma is als volgt.

We zetten het woordenboek op als een lijst met daarin lijstjes van twee elementen:

```
(SETQ wb '((spiritus geest)(quidem wel)(.....) .....)
```

De vertaalfuncties die we nodig hebben zijn van het Latijn naar het Nederlands (= LN) en omgekeerd (= NL):

```
(DEF (ln (woord wb)(COND
   ((NULL wb) NIL)
   ((EQ woord (CAAR wb))(CADAR wb))
   (T (ln (CDR wb))))))

(DEF (nl (woord wb)(COND
   ((NULL wb) NIL)
   ((EQ woord (CADAR wb))(CAAR wb))
   (T (nl (CDR wb))))))
```

• Varkenslatijn tot slot

Computermensen houden van spelletjes. In de text processing vindt men spelletjes als ELIZA, de woord voor woord vertaler, maar ook een spelletje dat een willekeurige taal vertaalt in de wereldtaal VARKENSLATIJN. De grammatica van het varkenslatijn is als volgt. Verwacht wordt een input string. Op deze input string worden de volgende transformaties toegepast om regulier varkenslatijn van het Nederlands te maken:

- zet van ieder woord in een zin de eerste letter achteraan;
- als het een klinker is, dan is het woord klaar;
- als het een medeklinker is, voeg dan IO toe aan het woord.

Voorbeeld: appyhio omputingcio.

Om dit probleem snel en slim te kunnen programmeren veranderen we de volgorde van de probleemstelling:

1. Blaas het eerstvolgende woord, dat nog niet behandeld is, op tot losse letters door middel van de functie EXPLODE.
2. Als de eerste letter van het aldus geëxplodeerde woord een klinker is, RETURN dan met COMPRESSie, nadat deze letter achteraan is geplaatst.
3. In alle andere gevallen hebben we te maken met een medeklinker; dan RETURNen we met een COMPRESS van de eerste letter en voegen daaraan toe de letters IO.

De klinkers leggen we vast onder de naam "klink":

```
(SETQ klink '(a, e, i, o, u, y)).
```

Dan definiëren we de hoofdfunctie die door middel van de standaard-functie MAPCAR werkt:

```
(DEF (vlatijn (zin)(MAPCAR zin 'vwoord)))
```

In de functie vlatijn vragen we herhaald de functie VWOORD toe te passen. Deze laatste krijgt van de MAPCAR telkens het eerstvolgende woord in zijn substitutie-variabele "woord" (zie de hierna volgende definitie van vwoord). Verder gebruiken we de PROGram feature, waardoor dit programma gaat lijken op ALGOL, BASIC, PASCAL, PL/I. Kortom, het gaat daarmee meer lijken op andere programmeertalen, zodat het gemakkelijker te lezen is. Binnen de PROG-haken leggen we de lokale variabelen lettw (van letterwoord) en woordv (van varkenswoord) vast. In lettw zetten we de geEXPLODEerde versie van ieder binnenkomend woord weg. De inhoud van lettw gebruiken we om in woordv alvast de omkering (= REVERSE) van de eerste letter (= CAR lettw) met de rest van de letters in lettw (= CDR lettw) vast te leggen. Daarna gebruiken we de conditionalizer (= COND) om te kijken of de eerste letter (= CAR lettw) soms MEMBER is van klink:

- zo ja, dan RETURN met de COMPRESS van woordv;
- in alle andere gevallen (= T) RETURN dan met de COMPRESS van het woordv samengevoegd met de letters io.

De gehele functie vwoord luidt in deze volgorde gedefinieerd:

```
(DEF (vwoord (woord)(PROG (lettw woordv)
   (SETQ lettw (EXPLODE woord))
   (SETQ woordv (REVERSE
                (CAR lettw)
                (CDR lettw)))
   (COND
   ((MEMBER (CAR lettw) klink)
   (RETURN (COMPRESS woordv)))
   (T (COMPRESS (APPEND woordv 'io)))))))
```

De tewerkstelling van VLATIJN gaat nu als volgt:

```
(vlatijn '(er was eens een kunsttaal))
```

Waarna de machine antwoordt met:

```
(re aswio ense ene unsttaalkio)
```

Het kan uiteraard anders, het kan zelfs in één regel, maar dan moeten er eerst meer eigenschappen van de programmeertaal LISP bestudeerd worden.

Het bovenstaande programma geeft een aardige indicatie hoe eenvoudig het programmeren in LISP kan zijn. Het probleem zelf is echter maar ten dele opgelost. We moeten voor een volledige oplossing de klinkers aa, ee, ij, oo etc. ook meenemen, terwijl de klinker y geen klinker is als hij voor andere klinkers voorkomt. In 'yoghurt' is de y geen klinker, maar weer wel in 'Kenya'.

Een versie van een volledig varkenslatijn-programma is dan:

1. Blaas het eerstvolgende woord, dat nog onbehandeld is, op tot een los-letterwoord (= lettw) door middel van de functie EXPLODE.
2. Als de eerste letter van lettw een y is of g e e n klinker is, RETURN dan met een COMPRESSie van lettw, nadat de eerste letter en de letters i en o in die volgorde achteraan gevoegd (= APPEND) zijn.
3. In alle andere gevallen hebben we te maken met één of meer klinkers aan het begin. Kijk of de tweede letter ook MEMBER is van klink; zo ja, RETURN dan met een COMPRESSie van de achteraan geAPPENDe eerste twee letters; zo nee RETURN dan met een COMPRESSie van de achteraan geAPPENDe eerste letter.

Het volledige programma luidt nu:

```
(SETQ klink '(a, e, i, o, u, y))
(DEF (vlatijn (zin)(MAPCAR zin 'vwoord)))
(DEF (vwoord (woord)(PROG (lettw woordv)
   (SETQ lettw (EXPLODE woord))
```

In het opgeblazen woord (= lettw) gaan we nu de volgorde verwisselen onder zekere voorwaarden. De voorwaarden zijn dat er geen uitzondering optreedt. Die uitzonderingen worden eerst opgespoord met behulp van de conditionalizer 'COND':

```
(COND
   ((OR
      (NOT (MEMBER (CAR lettw) klink))
      (EQ (CAR lettw) 'y))(RETURN
         (COMPRESS (APPEND (APPEND
                                 (CDR lettw)
                                 (CAR lettw))
                                 'io))))
   ((MEMBER (CADR lettw) klink)
      (RETURN (COMPRESS (APPEND (CDDR lettw)
         (APPEND (CAR lettw)
                  (CADR lettw))))))))
```

Dit programma geeft nu op (vlatijn '(eens was een taal)) de uitslag (nsee aswio nee aaltio). Ook namen zoals 'eysbergen' gaan correct: (vlatijn '(eysbergen ionisatie)) geeft (sbergeney nisatieio).

LITERATUUR

Berkeley, E.C. & D.G. Bobrow (ed.): "The programming language LISP: its operation and applications." M.I.T. Press, Cambridge, Mass., 1964.

Berry-Rogghe, G. & T.D. Crawford: "COCOA: A word count generator", Berkshire, Cardiff, 1973.

Boot, M.: "LIBOLIB, I", Utrecht, 1976.

Crenell, K.: "How to use COCOA to produce indexes?", ALLC Bulletin, 1975.

Hockey, S. & I. Marriott: "Oxford concordance program, Users' Manual", 1980.

Jones, A. & F. Churchhouse: "The computer in literary and linguistic studies", Berkshire, Cardiff, 1976.

Kaeding, F.: "Häufigkeitswörterbuch der deutschen Sprache", 1898.

Knuth, D.: "The art of computer programming", Vol. 3, Sorting and searching, Addison-Wesley, Reading, Mass., 1973.

Kucera, H. & W. Nelson Francis: "Computational analysis of present-day american English", 1967.

Meier, H.: "Deutsche Sprachstatistik", Hildesheim, 1967.

Nie, N.H. et al.: "SPSS: Statistical Package for the Social Sciences". McGraw-Hill, New York, 1975.

Nieuwenborg, E.: "Retrogade woordenboek van de Nederlandse taal", Antwerpen, 1969.

Reed, A.: "CLOC: A collocation package", ALLC Bulletin, 1977.

Rogers, C.: "Client centered therapy: current practice, implications and theory", 1951.

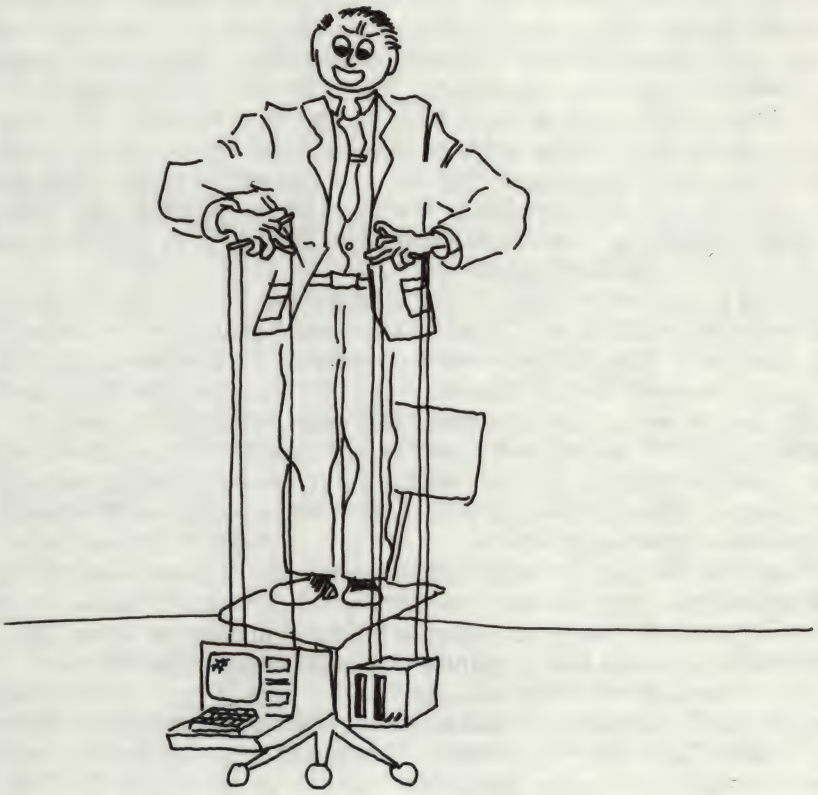
Siklóssy, L.: "Let's talk LISP", Prentice-Hall, Englewood Cliffs, New Jersey, 1976.

Uit den Bogaart, P.C.: "Woordfrequenties in geschreven en gesproken Nederlands", Utrecht, 1975.

Weizenbaum, J.: "ELIZA - A computer program for the study of natural language communication between man and machine", Comm. of the ACM, 9, 1966.

Winston, P.H.: "Artificial intelligence", Addison Wesley, Reading, Mass., 1977.

HOOFDSTUK 3
JANTJE ZAG EENS PRUIMEN HANGEN: ONTWIKKELEN VAN NIEUWE PROGRAMMATUUR VOOR TEXT PROCESSING



Drie aspecten zijn te onderscheiden aan taal- en text processing: een zoekaspect (het geheugenaspect), een denkaspect (het cognitieve aspect) en een uitdrukkingsaspect (het generatieve aspect). Onder het zoekaspect vallen veel van de functies die in hoofdstuk 2 werden behandeld. Alle vormen van boekhouden aan de hand van teksten dienen om gemakkelijk informatie terug te vinden in teksten. Hieronder vallen echter ook alle vormen van taalanalyse. Immers ook zinsontleden is een vorm van zoeken van informatie in een tekst. We noemen dit aspect dan ook 'handling linguistic information'. Handling linguistic information loopt van het simpel vervaardigen van indexen tot het ontleden van zinnen of het omzetten van geschreven taal in gesproken taal. Ook het vinden van verbanden tussen zinnen en zinsdelen behoort bij dit aspect van text processing.

Veel van het werk dat niet puur te maken heeft met de fysieke structuur van teksten moet nog gedaan worden. Wel zijn er verschillende voorstellen gedaan in de geschiedenis van de computerlinguistiek om tot oplossing van detailproblemen te komen. Een echte doorbraak op dit gebied is echter nog niet voorgekomen. Onder doorbraak verstaan we dan dat er oplossingen werden aangedragen die algemeen toepasbaar zijn en niet in hun toepasbaarheid alleen betrekking hadden op een (gedeelte van) een enkele taal. Nu is de wetenschap van taal en tekst zo opgebouwd dat de puur taalkundige kant nog het verst ontwikkeld is. Voor de verdere stappen omhoog naar de volledige tekst toe geldt zodoende de uitspraak nog sterker dat er niet echt algemeen toepasbare principes zijn gevonden die op allerlei soorten van 'inhoudelijke text processing' van toepassing zijn. Wel zijn er interessante voorstellen gedaan op deelgebieden.

Gaat men boven het woord- en/of zinsniveau uit dan stoot men bovendien op een extra moeilijkheid. Voor het behandelen en begrijpen van alinea's of hele teksten heeft men denkfuncties nodig die buiten het werk van woorden en zinnen begrijpen omgaan. Het is iets anders om de zin te ontleden: "Jantje zag eens pruimen hangen, o als eieren zo groot" dan de conclusie te trekken dat die pruimen kennelijk erg aanlokkelijk zijn voor Jantje, juist vanwege hun grootheid en daardoor verlokkende sappigheid. Om deze informatie, die in de zin zelf helemaal niet gegeven is, op te sporen moet men beschikken over een functie die gevolgtrekkingen kan maken op grond van allerlei soorten kennis. Hier kennis van de wereld, de pruim en de kinderziel in het bijzonder. Maar ook al op een lager niveau is meer dan puur taalkundige kennis nodig, tenminste taalkundig in de zin van: "we blijven binnen de grenzen van een zin", zoals in veel taalkundige studies geschiedt. Leest men namelijk na "Jantje zag eens pruimen hangen" de zin "Die moet ik hebben", dan heeft men een functie nodig die de verbinding legt tussen 'pruimen' uit de vorige zin en 'die' in deze zin. Het maken van verwijzingen in teksten blijkt een van de moeilijkste onderwerpen te zijn uit de zich moeizaam onder de pruimenboom voorttastende tekstwetenschap.

In de onderstaande paragrafen zullen de problemen en probleempjes één voor één worden aangeduid. Vooraf moet echter worden vastgesteld dat wel voor die aparte gebieden, zoals bijvoorbeeld het opmaken van conclusies die niet letterlijk in de tekst staan, aparte computerfuncties, ja aparte programmeertalen zijn gemaakt. Het is duidelijk dat we deze programmeertalen en functies niet alle aan de orde kunnen stellen in die zin dat ze ook direct bruikbaar worden. Wil men op het niveau van 'inferencing' werken, zoals het maken van impliciete gevolgtrekkingen, dan moet men kunnen beschikken over de programmatuur zelf (bijvoorbeeld het programma SAM, zie paragraaf 3.3) en met die programmatuur als uitgangspunt verder kunnen gaan experimenteren. In een boek als dit kunnen we dat niet bieden. Wel kunnen we een serie problemen en probleempjes aan de orde stellen en zodanige programmeermiddelen aanreiken dat die kunnen worden opgelost of dat men een aanwijzing krijgt naar de oplossing van de problemen. Dit laatste nu is de doelstelling van dit hoofdstuk, hetgeen we kunnen samenvatten met de zinsnede: aan de orde stellen van problemen die samenhangen met inhoudelijke text processing en zoveel mogelijk programmeertechnische middelen aanreiken om althans die problemen op te lossen die niet andere systemen vooraf vereisen.

Zodoende blijven veel pruimen in de boom hangen.

3.1 EERST DE MIDDELEN DAN DE MORAAL: PROGRAMMEERTALEN VOOR TEXT PROCESSING

Bij het werken met taal en tekst moeten dikwijls woorden of stukken tekst worden vervangen door andere. Of het nu gaat om standaardbrieven, het bijhouden van rapporten, het opmaken van een krantenartikel of het vertalen van een roman van Dostojewsky, steeds worden woorden of stukken tekst vervangen door andere. Ook wanneer men converseert zou men met een beetje goede wil kunnen spreken van het vervangen van de ene tekst door de andere. In ieder geval werkt ELIZA zo.

Nu bestaat de mogelijkheid dat er een één-op-één relatie is tussen wat vervangen moet worden en wat vervangt. Zo was de vervanging bij het varkenslatijn geregeld, zo is de vervanging geregeld wanneer men geschreven letters wil vervangen door hun gesproken (dat wil zeggen fonetische) vorm. Maar meestal is het zo dat de vervanging niet een op een is. De ene keer laat men hele stukken weg, de andere keer voegt men juist veel toe. Soms ontstaat er iets heel nieuws. Hieruit kan maar één conclusie getrokken worden ten aanzien van de programmeermiddelen die men moet aanschaffen om een han-

dige tekstmachine te krijgen: de programmeertaal (of -talen) moet(en) de programmeur in staat stellen met eenvoudige opdrachten patronen te herkennen en te vervangen door andere. Daarbij moet men niet worden gehinderd door de boekhouding die de computer zelf moet bijhouden als hij letters en reeksen letters moet verwisselen.

In de informatica stelt men zich doorgaans tevreden met programmeertalen waarin men 'in principe' op 'strings' kan werken. 'In principe' betekent dat men zelf eerst programma's moet gaan ontwikkelen om de boekhouding in de buik van de computer zelf te regelen. En 'strings' betekent dan niet meer dan, laten we zeggen, 'woordtekens', dus een serie letters waar een spatie omheen staat. We hebben boven al meer dan eens gezien dat er voor een tekstmachine wel wat meer komt kijken dan woorden alleen. Programmeertalen waarin men zelf moet gaan regelen hoe letters moeten worden opgeslagen, hoe woorden moeten worden gevonden, zijn voor de ontwikkeling van een tekstmachine niet echt interessant. Als er nu toevallig, zoals bij de SCANNER of bij de index-concordantie-aangelegenheid, standaardprogrammatuur te vinden is die dat soort taken al doet, is het verstandig die programma's te gebruiken. De ontwikkeling van nieuwe programmatuur echter moet worden toevertrouwd aan krachtiger programmeertalen: de problemen zelf zijn al ingewikkeld genoeg. Men hoeft niet bevreesd te zijn dat men werkeloos raakt of zijn creativiteit zal verliezen wanneer men een krachtige programmeertaal gebruikt.

SNOBOL

Programmeertalen waarmee patroonherkenning, wat dus meer is dan 'string handling', kan worden bedreven zijn SNOBOL, LISP en METEOR, om de paar te noemen die in dit boek worden geïntroduceerd. SNOBOL is daarvan de oudste en de minst doorzichtige.

De belangrijke voorloper van SNOBOL heette COMIT, de eerste taal waarmee echt patroonherkenning kon worden bedreven. COMIT zou later van doorslaggevend belang worden voor de programmeertaal METEOR, die in dit boek uitgebreid aan de orde gaat komen. Na enige jaren experimenteren met COMIT kwamen echter de gebreken naar voren. De moeilijkheid van COMIT was dat men strings niet kon benoemen en dat men evenmin enig rekenwerk ermee kon verrichten. In die jaren (d. w. z. de zestiger jaren) dacht men nog nauwelijks aan computertoepassingen met teksten waarin niet geteld hoefde te worden, dus men vond dat niet kunnen rekenen een groot bezwaar. Door de begeerte dat te verbeteren en door de behoefte ook een taal te krijgen die effectiever was dan COMIT begon een groep onderzoekers op de Bell Laboratories in 1962 te werken aan de eerste versie van SNOBOL. SNOBOL werd gedefinieerd als een 'string manipulation language'. Vooral op universiteiten werd deze nieuwe taal geliefd,

zonder dat er echter een werkelijke doorbraak onder grote gebruikersgroepen kwam. Het werkelijk belangrijke werk op het gebied van de kunstmatige intelligentie dat men kan beschouwen als een definitieve stap vooruit, werd niet gedaan in SNOBOL maar in LISP.

3.1.1 Nog een keer: groene ideeën: Een leuke toepassing van SNOBOL

Onder de invloed van Chomsky ontstond in Amerika van de vijftiger jaren een horde taalkundigen die meende dat de mens een apart soort functie of apparaatje in zijn hersenen had, waarmee hij zat uit te zoeken of zinnen wel netjes volgens de grammatica van zijn moedertaal gingen. Deze taalkundigen hadden het merkwaardige idee dat, wanneer je op straat een willekeurige neringdoende aanhoudt met de vraag of de zin "Jantje zag eens fietsen hangen" grammaticaal is, die neringdoende dan direct spontaan en braaf antwoordt dat die zin inderdaad grammaticaal is.

Nu is er geen empirisch onderzoek gedaan of de willekeurige neringdoende inderdaad dit antwoord geeft. Deze opvatting over het menselijke besef van grammaticaliteit is daardoor meer dan een soort dogma van volstrekte vanzelfsprekendheid gebleven. Het zal echter waarschijnlijker zijn dat men niet als antwoord de brave bevestiging van het dogma krijgt, maar eerder een antwoord waarin wordt getwijfeld aan de verstandelijke vermogens van de wetenschappelijke ondervrager. Desalniettemin zijn er in Nederland instituten voor algemene taalwetenschap waarin een (bijna) hooggeleerde zich onder- of bovenaan de trap bevindt met de klemmende vraag aan iedere binnenkomende student of Jantje met zijn hangende fietsen nu wel of niet grammaticaal is.

Het begrip grammaticaliteit nu, dat wil zeggen de strings die voorkomen in een bepaalde tekst moeten van een voorgeschreven vorm zijn, is wel van uitermate groot belang geweest voor het type p r o g r a m m e e r t a l e n dat zich onder invloed van Chomsky heeft ontwikkeld. Daarvoor en alleen daarvoor is het goed een testprogramma te hebben op grammaticaliteit. De ideale taal om een testprogramma op grammaticaliteit te schrijven is SNOBOL.

- BNF voor zinnetjes

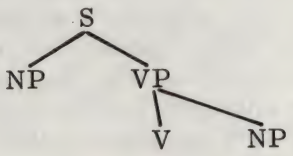
Een eenvoudig zinnetje in het Nederlands is bijvoorbeeld: "de linguist vangt een melkboer". Deze zin bestaat uit een onderwerp, een gezegde en een lijdend voorwerp. De zinsdelen zelf bestaan uit een of meer woorden, te weten een nominale groep (d.w.z. een groep woorden met een zelfstandig naamwoord erin), een verbale groep (d.w.z. een groep woorden met een werkwoord erin) en nog een nominale groep.

Grafisch aldus:

(de linguist (nominale groep (onderwerp)))
(vangt (verbale groep (gezegde)))
(een melkboer (nominale groep (lijdend voorwerp)))

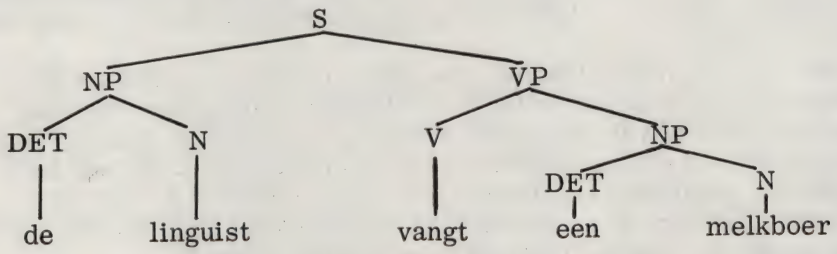
De structuur van deze zin wordt tegenwoordig bij voorkeur in een afhankelijkheidsdiagram (een zogenaamde binaire boom) weergegeven. Daarin worden de volgende symbolen gebruikt: S voor zin; NP (van 'Noun Phrase') voor nominale groep en VP (van 'Verb Phrase') voor verbale groep.

Het structuurdiagram ziet er dan als volgt uit:



De onderdelen NP en VP bestaan ieder weer uit andere onderdelen op lager niveau. In onze zin komen voor: lidwoorden ('de' en 'een'), zelfstandige naamwoorden ('linguist' en 'melkboer') en een werkwoord in de verbogen vorm ('vangt').

De aanduidingen voor die klassen van woorden zijn DET (van 'determiner') voor lidwoorden; V (van 'verb') voor werkwoord in de verbogen vorm en N (van 'noun') of SUBST (van substantief) voor zelfstandig naamwoord. Vullen we het structuurdiagram verder in, dan ziet het er als volgt uit:



Deze afhankelijkheidsboom nu kunnen we opvatten als een serie herschrijfsymbolen die loopt van het meest abstracte niveau (S) naar de meest concrete symbolen, de eindsymbolen of terminale symbolen, dat wil zeggen de woorden van de zin zelf. Daardoor is het diagram ook op te vatten als een serie herschrijfregels in een contextvrije grammatica. Dus is de boom op te schrijven in de notatie van Backus en Naur, de zogenaamde BNF-formalisering, zoals we die geleerd hebben in hoofdstuk 2.

De BNF-notatie ziet er dan als volgt uit:

```
<S>   ::= <NP><VP>
<NP>  ::= <DET><N>
<VP>  ::= <V><NP>
<DET> ::= ['de', 'een']
<N>   ::= ['linguist', 'melkboer']
<V>   ::= ['vangt']
```

Deze notatie van de afhankelijkheidsboom kan bijna regelrecht zo in SNOBOL worden opgeschreven. Het verschil is dat de BNF-formule moet worden omgekeerd, dat met andere woorden niet de S eerst wordt gedefinieerd, maar eerst de terminale symbolen. Verder moet men enige voorzieningen treffen dat de pointer (de aanwijzing voor de computer van waar af hij leest) aan het begin van de te lezen zin moet blijven staan en dat hij ook zijn bevind van zaken moet afdrukken. Ook moet er een opdracht komen om de input-zinnen te lezen en een opdracht wat te doen voor het geval er geen zinnen te lezen zijn. Dat alles levert voor de linguist met melkboer de volgende SNOBOL-tekst op.

• SNOBOL programma 1

```
&ANCHOR = 1     / dit is de pointer/
DET = 'DE' !  'EEN'     / het uitroepteken is de logische OR en
                          betekent 'of' /
N = 'LINGUIST' ! 'MELKBOER'
V = 'VANGT'
NP = DET N
VP = V NP
S = NP VP

LZ
        STRING = TRIM (INPUT)          : F (END)
                                  /dit is de leesopdracht: alles
                                  wat op een input-kaart staat wordt
                                  aan 'string' toegekend. De blanke
                                  posities na de laatste letter worden
                                  weggelaten.
                                  :F (END) betekent: als er niets
                                  meer te lezen is, ga dan naar END./

        OUTPUT = STRING     / druk de gelezen zin af./
        STRING S . PAT          :F (FOUT)
                                  / deze opdracht past de patroonher-
                                  kenning toe, d.w.z. deze opdracht
                                  doet de grammaticale analyse vol-
                                  gens de herschrijfregels./
```

```
        OUTPUT = 'STRING IS GRAMMATICAAL'          : (LZ)
FOUT
        OUTPUT = 'STRING IS NIET GRAMMATICAAL'     : (LZ)
                                / komt de machine hier dan was de
                                zin niet ontleedbaar, d.w.z. toe te
                                kennen aan het gedefiniëerde
                                patroon en dus is de zin niet gram-
                                maticaal, wat dan ook wordt
                                gedrukt. Daarna keert de machine
                                terug naar punt LZ, d.w.z. de vol-
                                gende zin wordt gelezen.//

END
```

- Een kleine uitbreiding van de grammatica

De contextvrije grammatica die we boven behandeld hebben is wel erg klein. Daarin zijn alleen maar zinnetjes mogelijk waarin een lidwoord wordt gevolgd door een zelfstandig naamwoord, wordt gevolgd door een werkwoord, wordt gevolgd door een lidwoord, wordt gevolgd door een zelfstandig naamwoord. Nu is het zo dat in het Nederlands tussen lidwoorden en zelfstandige naamwoorden ook nog andere woorden kunnen voorkomen, bijvoeglijke naamwoorden bijvoorbeeld en bijwoorden.

Voorbeeld: De uiterst slimme linguist vangt een nogal aardige melkboer.

Om deze zin te kunnen vangen in een BNF-notatie moeten we de definitie van NP uitbreiden. We moeten daarin bijvoorbeeld opnemen dat een NP ook kan bestaan uit de sequentie lidwoord-bijvoeglijk naamwoord-zelfstandig naamwoord. Iedereen die Nederlands kent weet bovendien dat ook meer dan één bijvoeglijk naamwoord achter elkaar kan worden gebruikt. Wat van belang is bij een NP is het zelfstandig naamwoord (de N), de rest is veel vrijer. Zodoende krijgt men iets vervelends voor een abstracte notatie als de BNF: men zou in principe een volledige opsomming moeten geven, terwijl men van tevoren weet dat er toch altijd weer een bijvoeglijk naamwoord of een bijwoord bij kan. Het verdient aanbeveling een eenvoudiger manier van noteren uit te vinden.

Deze eenvoudiger manier van noteren bereikt men door in de notatie de NP zichzelf te laten aanroepen voor die stukken die herhaalbaar zijn. Men noemt dat r e c u r s i e v e a a n r o e p of recursieve notatie.

Stel nu dat men NP heeft gedefinieerd als de opeenvolging van bijvoeglijk naamwoord en zelfstandig naamwoord: NP ::= ADJ N en dat men daaraan wil toevoegen: NP ::= DET ADJ N, dan kan

men ook schrijven: NP ::= DET NP. Daarmee roept NP zichzelf opnieuw aan en in feite staat er dus:

NP ::= ADJ N ! DET ADJ N

In SNOBOL geeft men bij een recursieve definitie in de grammatica voor het symbool dat recursief wordt gedefinieerd een * op. De regel SNOBOL voor de boven recursief gedefinieerde NP zou dus als volgt moeten worden geschreven:

NP ::= ADJ N ! DET *NP

Deze kennis is voldoende voor de bestudering van de volgende programmatekst.
(Opmerking: In plaats van het symbool N voor zelfstandig naamwoord wordt in het programma het symbool SUB gebruikt. N = de afkorting van 'nomen', het Latijnse woord voor naamwoord en SUB = de afkorting van 'substantivum', het Latijnse woord voor zelfstandig.)

SNOBOL programma 2

```
          &ANCHOR = 1
        DET = 'EEN ' ! 'DE ' ! 'HET '
        PP = 'IK ' ! 'JY ' ! 'HY '
        SUB = 'KAT ' ! 'HOND ' ! 'VIS '
        ADJ = 'OUDE ' ! 'JONGE ' ! 'GLIMMENDE '
        VB = 'GA ' ! 'ZIE ' ! 'ZIET '
        NP = ADJ SUB ! PP ! DET *NP ! DET SUB
        VP = VB NP ! VB
       ZIN = NP VP ! VP
LZ
          STRING = TRIM(INPUT)          :F(END)
          STRING = STRING ' '
           OUTPUT = STRING
          STRING ZIN . PAT              :F(FOUT)
         OUTPUT = 'STRING IS GRAMMATIKAAL'        :(LZ)
FOUT
           OUTPUT = 'ZIN NIET GRAMMATIKAAL'    :(LZ)
END
```

VRAGEN EN OPGAVEN naar aanleiding van het SNOBOL programma 2, ter programmering van een contextvrije grammatica voor het Nederlands.

1. Bestudeer de contextvrije grammatica voor het Nederlands in SNOBOL.
2. Welke van de volgende zinnen accepteert het programma als grammaticaal?
   a. ik zie een kat
   b. ik oude
   c. kat hond oude zie
   d. ik zie een glimmende vis
   e. de hond ziet een jonge kat
   f. ga oude honde
   g. ik een oude hond zie
3. Vorm zelf een paar zinnen op grond van de input van het programma die ook als grammaticaal zouden worden geaccepteerd.

- Het nut van CF-grammatica's en waarom SNOBOL niet verder nodig is

Het is zonder meer duidelijk dat SNOBOL een machtig instrument is voor het noteren en met de computer uittesten van contextvrije grammatica's. De vraag naar het nut van deze vorm van grammaticale notatie klemt nu dus des te meer. Immers, mocht deze vorm van grammaticale beschrijving van een taal belangwekkend zijn, dan is met SNOBOL direct een superieur instrument voor de taalkundige gevonden. Het nadeel van de CF-notatie (Context Free) ziet men weldra in wanneer men probeert 'alle' zinnen te vinden die onze grammatica goed vindt. Bijvoorbeeld:

De de oude vis zie het de glimmende hond.

Deze zin zou volgens onze definities worden geaccepteerd. Nu hebben we alleen nog maar een grammaticaatje gemaakt voor enkelvoudige korte zinnetjes in het Nederlands. Deze zinnetjes zijn bovendien nog niet eens anders dan gewoon mededelend. De vraagzin en de uitroepzin ontbreken. Ook tussengevoegde zinnen ontbreken. Maar toch hebben we nu al grote moeite om te blijven overzien wat er precies allemaal geaccepteerd zal worden door de grammatica als goed Nederlands. Deze onoverzichtelijkheid zal alleen nog maar toenemen wanneer we complexere zinnen toelaten. Complexere zinnen (complex gedefinieerd vanuit de CF-grammatica!) zijn heel gewoon in het Nederlands.

Ofwel: CF-grammatica's zijn goed voor heel eenvoudige taaltjes, waarin het aantal input strings redelijk te overzien is. Ze zijn zodoende redelijk geschikt voor computertalen; ongeschikt echter

voor behoorlijke menselijke talen. Daarbij komt de kwestie die we boven al aan de orde stelden: dat de vraag of een zin nu al of niet grammaticaal is niet zo erg ter zake doet in het proces van taalbegrijpen. Daar spelen heel andere zaken een rol, en een normale Nederlander is best in staat het zinnetje: "Mamma, kous" van een kleuter of "Ikke niet verstaan" van een Turk te begrijpen. Het is jammer, maar dit grote voordeel van SNOBOL is niet erg belangrijk voor de tekstmachine.

3.1.2 Vleeseters, woordvelden en de weg omhoog

In paragraaf 2.3.2 werd aangeduid dat voor een echte index op onderwerp de noodzaak bestaat om woordvelden te maken met de computer. Het is minder aanbevelenswaardig alle informatie met de hand in de computer aan te brengen. Een klassiek voorbeeld van de organisatie van woordvelden is het classificatiesysteem dat door Linnaeus werd bedacht om levende wezens in onder te brengen.

Levende wezens worden volgens dit systeem ondergebracht in grote groepen. Deze grote groepen worden dan weer verder onderverdeeld op bepaalde eigenschappen. Een dergelijk schema van classificatie noemt men hiërarchie. Zoals we boven al zagen bij de structuurboom van een zin in het Nederlands is het denken in hiërarchische structuren een vriendelijke manier om de computer te behandelen. Het is vrij gemakkelijk die structuren om te zetten in voorstellingen die de computer kan behandelen.

De hiërarchie wordt dus bepaald door eigenschappen. De eigenschappen die bij elkaar horen plaatst men in lijsten. Zo verkrijgt men 'eigenschappenlijst-structuren om het maar eens heel lelijk te zeggen ('property-list structures' is de vakterm hiervoor).

Tot nu toe werden hiërarchieën abstract weergegeven in BNF-format. Dit format kunnen we toepassen op levende wezens, en wel als volgt:

```
<DIER>          ::= <GEWERVELD> <ONGEWERVELD>
<GEWERVELD>     ::= <VOGEL> <VIS> <AMFIBIE> <ZOOGDIER>
<ONGEWERVELD>   ::= <WORM> <INSEKT>
<INSEKT>        ::= <MIER> <MUG> <SPIN>
<VOGEL>         ::= <ADELAAR> <PINGUIN> <VINK>
<VIS>           ::= <SCHOL> <HARING> <DOLFIJN>
<AMFIBIE>       ::= <KIKKER> <HAGEDIS>
<ZOOGDIER>      ::= <PAARD> <HOND> <PRIMAAT> <VLEESETER>
<VLEESETER>     ::= <HOND> <KAT>
<PRIMAAT>       ::= <AAP> <MENS>
```

Opmerking: Nauwkeurige bestudering van het lijstje zal opleveren dat niet alle eigenschappen van gelijk niveau bij elkaar staan. Zo is de

categorie 'paard' niet van dezelfde orde als de categorie 'vleeseter'. Dit doet echter aan het principe niets af.

Met een dergelijke ordening kunnen we minstens twee dingen doen. We kunnen indexen maken door te zoeken welke eigenschappen bij een bepaald woord horen. Stel dat een index moet worden gemaakt op het geheime sexleven van de dieren in Nederland, dan is het met een dergelijke property-list structuur mogelijk 'hond' en 'paard' onder te brengen bij de verwijzingen naar zoogdieren (de weg omhoog door de boom).

De tweede toepassing is: vragen door de computer laten beantwoorden. Zo zal de computer nu kunnen vertellen dat een hond een vleesetend, gewerveld zoogdier is en dat hij dat gemeen heeft met een kat.

Dit is te leuk om niet onmiddellijk te programmeren. Dit programmeren gaat in de lijsten-taal bij uitstek: LISP. Voor we daar echter toe overgaan moet de titel van deze hoofdparagraaf (eerst de middelen dan de moraal) nog worden ingelost. Het zal duidelijk zijn dat teksten voor een groot deel bestaan uit lijstenwerk. Lijsten betekent nu echter niet meer het maken van woordenlijstjes, maar lijsten van dieper en intelligenter niveau. Lijsten van het soort waarmee je BNF-spelletjes kunt spelen. Het best kan dat met een computertaal die ook zelf uit lijsten bestaat. Die taal is LISP. Ofwel om nieuwe instrumenten te maken ter bespeling van de tekstmachine gebruikt men het beste LISP. Het is namelijk moreel verantwoord het beste instrument te gebruiken, niet alleen wanneer men een spijker in de muur slaat, maar ook wanneer men denkwerk moet verrichten.

Aan een keuze voor LISP zou op het eerste gezicht echter een bezwaar kunnen kleven. Dat bezwaar is dat men ook gaat 'lijsten' en 'bomen' waar het niet nodig is. In teksten gebeurt veel op pure patroonherkenning. We willen het beste instrument hebben, dus moeten we ons afvragen of LISP patroonherkenning toestaat, zonder dat men onnodige omwegen hoeft te maken over bomen en lijsten. Die vraag kan bevestigend beantwoord worden. In LISP kan men namelijk gemakkelijk functies schrijven die heel andere opgaven verrichten. Die functies kan men dan weer als aparte taal gebruiken. Omdat de functie in LISP geschreven is, kan men de taal bovendien inbedden in LISP. Moet men dan de lijst-structuur en de hiërarchie gebruiken van 'gewoon' LISP, dan staan die onmiddellijk klaar. Nu is met de zinsnede "in LISP kan men gemakkelijk functies schrijven" niet bedoeld dat men dat ook zelf moet gaan doen. Integendeel, die functies zijn kant en klaar op te halen uit de literatuur. Een beroemde patroonherkennende taal in LISP heet METEOR. Deze taal zullen we dan ook gebruiken voor die zaken die het best via patroonherkenning kunnen worden opgelost. Ofwel: bij gebruik van LISP hoeft men niet telkens opnieuw het programmeerwiel te gaan zitten uitvinden, men kan denken over meer inhoudelijk geformuleerde problemen.

En nu zijn we klaar om werkelijk te gaan programmeren.

3.1.2.1 Met LISP in de aap gelogeerd: Een data-base

De BNF over dieren wordt in LISP geprogrammeerd. We hebben de feiten al genoemd in Backus-Naur-notatie en we leggen deze nu vast in de computer met behulp van LISP. De standaardoperatie voor vastleggen van zaken is in LISP-taal de SETQ operatie. We leggen de feiten vast onder de naam DATA_BASE, als volgt.

```
(SETQ DATA_BASE '(
(DIER GEWERVELD ONGEWERVELD)
(GEWERVELD VOGEL VIS AMFIBIE ZOOGDIER)
(ONGEWERVELD WORM INSEKT)
(INSEKT MIER MUG SPIN)
(VOGEL ADELAAR PINGUIN VINK)
(VIS SCHOL HARING DOLFIJN)
(AMFIBIE KIKKER HAGEDIS)
(ZOOGDIER PAARD HOND PRIMAAT VLEESETER)
(VLEESETER HOND KAT)
(PRIMAAT AAP MENS)
))
```

Uit een gegevensbestand (= data-base) wil een gebruiker feiten halen en hij wil er feiten aan toe kunnen voegen. De functie VOEG brengt nieuwe feiten in het bestand DATA_BASE door deze vooraan te plaatsen, maar hij kijkt wel eerst of dit feit er soms al in zat.

```
(DEF (VOEG (NIEUWFEIT)(COND
   ((MEMBER NIEUWFEIT DATA_BASE) NIL)
   (T (SETQ DATA_BASE (CONS NIEUWFEIT DATA_BASE)))
)))
```

De functie VEEG haalt oude feiten uit het bestand DATA_BASE door eerst te kijken of deze er wel in zitten.

```
(DEF (VEEG (OUDFEIT)(COND
   ((MEMBER OUDFEIT DATA_BASE)
      (SETQ DATA_BASE (EFFACE OUDFEIT DATA_BASE)))
   (T NIL)
)))
```

OPGAVE. Vereenvoudig de VEEG-functie in belangrijke mate.

Nu kunnen we feiten toevoegen of wegvegen, maar nog geen vragen stellen aan de DATA_BASE. Dat gaan we doen door bemiddeling van de functie VRAAG. Deze KIJKt of de desbetreffende woorden in de gegevens zitten en geeft dan het bijbehorende feit weer.

```
(DEF (VRAAG (WOORD GEGEVENS)(COND
   ((NULL GEGEVENS) NIL)
   ((KIJK WOORD (CAR GEGEVENS))
      (CAR GEGEVENS))
   (T (VRAAG WOORD (CDR GEGEVENS)))
)))
```

Het is zaak nu de functie KIJK te definiëren.

```
(DEF (KIJK (PATROON GEGEVEN)(COND
   ((NULL PATROON) NIL)
   ((NULL GEGEVEN) NIL)
   ((MEMBER PATROON GEGEVEN) T)
   (T NIL)
)))
```

Het WOORD uit de VRAAG-functie vormt het PATROON voor de KIJK-functie. Er wordt in KIJK gekeken of dit PATROON in het door middel van (CAR GEGEVENS) opgeleverde GEGEVEN zit; zo ja dan wordt True geleverd aan de VRAAG-functie die dan weet dat de desbetreffende GEGEVENS relevant zijn. VRAAG levert die dus op als antwoord op de vraag. In zijn boek "Artificial Intelligence" geeft Winston (zie literatuurlijst) zich veel moeite de hier gehanteerde MEMBER te programmeren als een MATCH-functie die een zwakke afschaduwing is van hetgeen Bobrows METEOR kan. Dat laatste wordt uitgebreid in dit boek behandeld. De hier gehanteerde MEMBER is in de meeste LISP-systemen aanwezig en zoekt of PATROON ergens in GEGEVEN te vinden is.

Het stellen van vragen aan deze data-base gaat nu bijvoorbeeld met (VRAAG 'SPIN DATA_BASE), het antwoord is dan (INSEKT MIER MUG SPIN). Een andere vraag, bijvoorbeeld naar honden: (VRAAG 'HOND DATA_BASE) levert het antwoord (ZOOGDIER PAARD HOND PRIMAAT VLEESETER). En zo kan dat.

Een aardiger toepassing van dit gegevensbestand is het trekken van conclusies. Bijvoorbeeld: Men vraagt aan de computer om gegevens over de spin en deze antwoordt met (EEN SPIN IS INSEKT EN ONGEWERVELD). Ander voorbeeld: Men vraagt aan de computer om gegevens over de mens en er komt dan: (EEN MENS IS PRIMAAT EN ZOOGDIER EN GEWERVELD).

Door het vraagwoord eerst te zoeken, te CONSen met 'een' en dan terug te lopen naar het eerste woord van die regel, te CONSen met 'is' wordt deze output uit de boom geconstrueerd.

OPGAVE. Bestudeer de data-base voor dieren uit het nieuwe boek van Winston en Horn, getiteld "LISP" (zie literatuurlijst).

• Klauwen en krulstaarten: Het toevoegen van eigenschappen

Opgemerkt werd al in de voorgaande paragraaf dat allerlei dieren in het woordveld naast en bij elkaar stonden, terwijl er toch duidelijk extra verschillen waren. Om tot een preciezere indeling te komen zouden we graag extra eigenschappen willen toevoegen. Zo behoren vissen wel tot de gewervelde dieren, maar ze zijn toch verschillend van zowel de vogels als de amfibieën als de zoogdieren door het feit dat ze koudbloedig zijn e n tegelijkertijd altijd in het water leven (wanneer we tenminste even afzien van de slijkspringer, een vis die ook op het land leeft; dit is mogelijk doordat hij op geregelde tijden even naar het water teruggaat om een voorraadje water voor zijn kieuwactiviteit in te nemen). Om deze specifiekere informatie op te nemen gebruiken we onderlijsten ('sublists'), als volgt (we gebruiken nu een haakjesnotatie en geen BNF):

```
DIER       = ((gewerveld) (ongewerveld))
GEWERVELD  = ((vogel) (vis) (amfibie) (zoogdier))
VOGEL      = (((adelaar) (pinguin) (vink)) (kan vliegen)
             (warmbloedig))
VIS        = (((schol) (haring) (dolfijn)) (leeft in het water)
             (koudbloedig))
AMFIBIE    = (((kikker) (hagedis)) (leeft op het land) (koudbloedig))
ZOOGDIER   = (((paard) (hond) (primaat) (vleeseter)) (woont op het
             land) (warmbloedig))
VLEESETER  = (((hond) ((kat) (heeft klauwen))) (kleine herseninhoud))
PRIMAAT    = (((aap) (mens)) (heeft nagels) (grote herseninhoud))
```

Er zijn nu eigenschappen aangegeven die bij de hele groep horen, maar ook eigenschappen die slechts bij één der genoemde dieren horen. Zo hoort de eigenschap 'op het land wonen' bij zowel de amfibieën als de zoogdieren. De eigenschap 'heeft klauwen' hoort alleen bij de kat. De krulstaart zou men moeten toevoegen bij de warmbloedige, op het land wonende alleseter: varken.

We hebben dus nu een eigenschappenlijst, een property list. Een nog mooier woord voor deze lijststructuur is s e m a n t i s c h n e t w e r k .

De extra informatie die toegevoegd is, zoals 'leeft op het water', 'leeft op het land', etc. geeft letterlijk eigenschappen (= properties) weer van vissen en amfibieën. Deze eigenschappen spelen geen rol in de Backus-Naur-formulering, zij behoren niet tot de hiërarchieën van afleidingsregels, daarom verlieten we de BNF-notatie. Deze properties nu kunnen in LISP gemakkelijk in het netwerk gehangen worden met behulp van de property-lists. Dat gaat als volgt:

```
(PUT 'SCHOL 'LEEFMILIEU '(LEEFT IN HET WATER))
(PUT 'HARING 'LEEFMILIEU '(LEEFT IN HET WATER))
(PUT 'DOLFIJN 'LEEFMILIEU '(LEEFT IN HET WATER))
(PUT 'SCHOL 'BLOED 'KOUDBLOEDIG)
(PUT 'HARING 'BLOED 'KOUDBLOEDIG)
```

Hier gebeuren onaangekondigde dingen. LISP vraagt namelijk de op te nemen eigenschappen van de schol, de haring, etc. te voorzien van aanwijzers, de zogenaamde indicatoren. Voor deze vissen zijn dat het leefmilieu en het bloed. Dat mogen er veel meer zijn. De functie PUT is zo gemaakt dat er, voor er een eigenschap toegevoegd wordt, eerst gezocht wordt naar de gemelde indicator (in de bovenste regel: 'leefmilieu') of deze soms al voorkwam op de indicatorenlijst van de 'schol' en zo ja dan wordt de nieuwe eigenschap 'leeft in het water' onder de reeds bekende indicator opgehangen. Nu is vroegere informatie vervangen. Als de genoemde indicator 'leefmilieu' niet gevonden wordt, plaatst PUT een nieuw element erbij, in dit geval LEEFMILIEU met de eigenschap '(LEEFT IN HET WATER). Zo kan er willekeurig veel informatie in een netwerk opgehangen worden.

Het ophalen van eigenschappen van de schol gebeurt eenvoudig met:

```
(GET 'SCHOL 'LEEFMILIEU)
(GET 'SCHOL 'BLOED)
```

En dan komt er respectievelijk 'leeft in het water' en 'koudbloedig'.

Om nu te kunnen vragen naar bijvoorbeeld alle koudbloedige dieren die in het water leven moet er patroonherkenning op de eigenschappen gedaan worden door deze laatste eerst op te halen. Een dergelijke taak gaat met (EQUAL '(LEEFT IN HET WATER) (GET DIERNAAM 'LEEFMILIEU)) en (EQUAL 'KOUDBLOEDIG (GET DIERNAAM 'BLOED)), waarbij voor 'diernaam' iedere diernaam uit de data-base ingevuld mag worden. Dan komt er:

```
(DEF (VRAAG2 (DIERNAAM)(COND
   ((AND
      (EQUAL '(LEEFT IN HET WATER)
             (GET DIERNAAM 'LEEFMILIEU))
      (EQUAL 'KOUDBLOEDIG
             (GET DIERNAAM 'BLOED)))
    DIERNAAM)
   (T NIL))))
```

Om alle namen van dieren uit de data-base te krijgen die in het water leven en koudbloedig zijn moet deze vraagfunctie over de hele data-base gehaald worden. Dat kan eenvoudig met de functie MAPCAR. (Deze functie wordt verderop in dit boek uitgelegd.)

```
(MAPCAR DATA_BASE '(LAMBDA (X)(MAPCAR X 'VRAAG)))
```

Voor dit soort semantische netwerken zijn drie soorten toepassingen te vinden.

- Toepassingen van semantische netwerken

Op de eerste plaats geeft het semantische netwerk een mooi houvast wanneer men een overzicht wil hebben van de ordening van een bepaald gebied van kennis dat bestaat uit vele woordvelden. Zo zou men in het bovenstaande kunnen spreken van het woordveld dieren, maar ook van het kennisgebied dieren dat bestaat uit de woordvelden amfibieën, gewervelde dieren, etc.

OPGAVE. Maak de BNF voor dieren af als semantisch netwerk door ook voor de nog niet verder gedefinieerde delen sublists van specifieke eigenschappen toe te voegen.

Op de tweede plaats kan men, nadat het netwerk in de computer is ingebracht, vragen gaan stellen over de gegevens die in het netwerk zijn opgeslagen. We gaven daarvan al een voorbeeld.

OPGAVE. Schrijf een programma in LISP waarmee kan worden gevraagd welke dieren zowel op het land leven, als warmbloedig zijn.

Op de derde plaats kan men de computer conclusies laten trekken over kennis die niet expliciet in het netwerk (de data-bank) vermeld staat. Dit wordt aangeduid met de vakterm 'inferenties maken' ('inferencing'). Het automatisch trekken van conclusies, zogenaamd 'automatic inferencing' kan bevorderd worden door de VOEG- en VEEG-functies uit te breiden.

In de praktijk wordt er vaak informatie aan een gegevensbestand toegevoegd die erg dicht tegen reeds ingebrachte informatie aan ligt, maar die door MEMBER niet herkend wordt als OUDFEIT. Daardoor wordt bijvoorbeeld een NIEUWFEIT als (VIS MAKREEL SCHOL HARING DOLFIJN) toegevoegd terwijl het OUFEIT (VIS SCHOL HARING DOLFIJN) er ook in blijft zitten. De functies die bij de database behoren, speciaal de VOEG- en VEEG-functies zouden uitgerust moeten zijn met automatische boekhoudprocedures om dit soort informatie te vermijden. In de praktijk van 'intelligente' gegevensbestanden bouwt men die dan ook in.

Een eerste stap hiertoe is het inbouwen van een redelijk slimme VEEG-functie als er nieuwe feiten toegeVOEGd worden. Omdat meer gebruikers informatie zullen toevoegen komt een gegevensbestand waarop veel gewerkt wordt daardoor min of meer tot 'leven'. Het toevoegen van gegevens door de ene gebruiker maakt op een andere vrager van informatie een 'levendige' indruk als hij merkt dat binnen een sessie de feiten veranderd zijn. We breiden de VOEG-functie uit. Natuurlijk blijft de afzonderlijke VEEG-functie bestaan: het moet mogelijk blijven te vegen zonder iets te moeten toevoegen.

```
(DEF (VOEG2 (NIEUWFEIT)(COND
   ((VEEG2 (SIMILARITY NIEUWFEIT DATA_BASE))
    (SETQ DATA_BASE (CONS NIEUWFEIT DATA_BASE)))
   (T (SETQ DATA_BASE (CONS NIEUWFEIT DATA_BASE)))
)))
```

Daartoe moet ook de VEEG-functie vernieuwd worden.

```
(DEF (VEEG2 (OUDFEIT)(SETQ DATA_BASE
   (EFFACE (SIMILARITY OUDFEIT DATA_BASE) DATA_
                                          BASE))))
```

OPGAVE. Vereenvoudig de functie VOEG2.

Let op dat een nieuw toe te voegen feit voor de VEEG-functie een oud feit vormt, dat zo mogelijk geveegd moet worden. Voor dat vegen zorgt de standaardfunctie EFFACE.

Een aardigheidje is de nieuwe taak SIMILARITY, die kijkt of de nieuw in te brengen feiten en gegevens voldoende gelijkluidend zijn met reeds aanwezige data in het bestand. Zo ja, dan wordt dit nieuwe feit aan VEEG2 gegeven die dat als oud nieuws beschouwt en daarmee gelijkluidende informatie verwijdert. De taaknaam SIMILARITY is een idee van Terry Winograd, alias Tony Earwig. Een programmafragment van zijn hand wordt verderop in dit hoofdstuk gegeven. Hij gebruikt in zijn vragen-beantwoorder ANSQUEST de functie PLAUSIBILITY die zo'n soort rol vervult als hier SIMILARITY. De taak SIMILARITY is te construeren met Bobrows patroonherkenner

in LISP: METEOR. METEOR is gemakkelijk te hanteren, maar volgt pas later in dit hoofdstuk.

Het feit dat er in dit boek zo'n sophisticated patroonherkenner als METEOR aanbevolen wordt, betekent nog niet dat daarmee alle ideeën over w a t herkend moet worden om SIMILARITY te bouwen onmiddellijk duidelijk zijn. Daarom daarover iets meer. Het ligt immers al in het woord 'similarity' besloten: de mate waarin nieuwe informatie op oude gegevens lijkt, is geen kwestie van klassieke ja-nee logica. Integendeel, een goede maat voor similarity onderscheidt vele gevallen tussen de klasse ja en nee, true en false. Een eigentijdse maat hiervoor is te vinden in de moderne 'fuzzy set' theorie, dat is een poging van de wiskundige Zadeh om de mate van gelijkenis (= similarity) op strenge wijze te hanteren en daardoor geschikt te maken voor computertoepassingen. Informatie-ontsluiting zoals we hier deden: met behulp van de functie MEMBER, die eigenlijk een klassieke predicaatfunctie is, kan dus ook met de 'mate van lidmaatschap', dat is 'degree of membership'.

3.1.2.2 Informatie-ontsluitende logica

In het algemeen kan een proces voor ontsluiting van informatie gezien worden als een behandeling van een verzoek om informatie uit een bestand aan informatie. In de zogenaamde beschrijvende systemen ('Descriptor Systems') wordt de te vinden informatie ontsloten door te vergelijken met de beschrijving van de inhoud van het databestand. De voorbereiding van deze beschrijving wordt meestal 'indexering' genoemd. Indexering is de toewijzing van een aantal beschrijvende woorden aan het databestand om daarmee zowel een aanduiding van de inhoud te verkrijgen als ook een middel om de voorhanden informatie op te halen. Het criterium van voorhanden informatie is dat de beschrijving van de informatie voldoende overeenkomt met de beschrijvingen in het verzoek. In LISP heten de indexen 'indicatoren'. Als zodanig is indexering handig, want zoeken naar indexen of indicatoren gaat ook voor computers sneller dan zoeken naar informatie-zelf.

Als b een beschrijving is en x is een informatie-eenheid, neem dan de uitspraak 'x heeft b' eens onder de loupe. We symboliseren deze uitspraak met de enkele letter p, van predikaat. Dan kunnen we de manieren waarop we een verzoek om informatie formuleren, heel sterk vereenvoudigen door het verzoek te vatten in een formule die opgebouwd is met de verbindingswoorden 'and', 'or' en 'not'.

In een tweewaardige-uitspraken-systeem moeten alle uitspraken de waarden 'waar' of 'onwaar' hebben. Daardoor ontstaan voor dit soort uitspraken systemen (ook wel uitspraken-logica's), de zogenaamde 'waarheidstabellen.

Met het gebruik van waarheidstabellen kunnen alle uitspraken, hoe complex ook, in een tweewaardig uitsprakensysteem berekend worden. Een eenvoudig voorbeeld is de formulering 'p and q'. Als zoekopdracht staat hier: geef die informatie waarvoor de beschrijving p and q waar is. Het is duidelijk dat er nu vier combinaties van waarheidswaarden voor p en q optreden. Waarbij 'waar' genoteerd wordt met '1' en 'onwaar' met '0'.

p	q	p and q
0	0	0
0	1	0
1	0	0
1	1	1

Als tweewaardige uitspraken gehanteerd worden voor de onderliggende logica van een informatieverwerkend systeem, dan is de Algebra van Boole een goed model voor dit proces. Met nadruk zij gezegd, dat hier de waarheidswaarde die een zoekopdracht oplevert na behandeling gelijk is aan de mate van relevantie die de ontsloten informatie heeft ten aanzien van de gezochte informatie.

De belangrijkste vraag is nu: wat is het nut van die zogenaamde Boolse algebra in deze? Om een goede zoekvraag te kunnen doen moeten de beschrijvingen voor informatie, de zogenaamde 'indicatoren', goed begrepen worden. Een beschrijving is een inhoudaangevend woord of zin, die de gevraagde informatie samenvat. Uiteraard is een indicator van een tekst een benadering van de inhoud van die tekst. Deze samenvattingen worden nog steeds door mensen gemaakt. Mensen coderen de informatie in teksten zo dat de samenvattende uitspraken een benadering vormen van de oorspronkelijke inhoud. Daarom kan de logica voor het indexeren niet uitsluitend de Boolse zijn, maar moet een graduering van benadering van de oorspronkelijke inhoud kunnen weergeven, dus moet het een fuzzy logica zijn. De diepere achtergrond hiervan is dat met de tweewaardige (Boolse) manier van redeneren het ontsluitingssysteem die antwoorden geeft waarvoor de zoekopdracht de waarde 1, dat is 'waar' heeft. Met het gebruik van een fuzzy redeneringssysteem worden die gegevens ontsloten die op bepaalde wijze met de zoekspecificatie overeenkomen. Deze benadering 'op bepaalde wijze overkomen' heet ook wel 'approximate reasoning' (zie Kickert, Koppelaar, Negoita en Wenstøp uit de literatuurlijst) en behelst uit de aard der zaak een benadering met fuzzy (= vage) logica. Daarom kan de logica voor indexering niet de klassieke tweewaardige zijn.

Met de Boolse tweewaardige algebra levert een gegevensbestand slechts die informatie waar 'letterlijk' om gevraagd wordt. Met het fuzzy model komen er gegevens los die op een of andere manier overeenkomen met de vragen in kwestie. Bijvoorbeeld met de formule

p and q krijgen we al oneindig veel 'waarheidswaarden'. Dat komt doordat p en q op zich al iedere waarheidswaarde tussen de getallen 0 en 1 kunnen aannemen, dus 0.3, 0.341, 0.4, 0.57, etc.

p	q	p and q
0.8	0.7	0.7
0.7	1.0	0.7
0.6	0.3	0.3
0.5	0.2	0.2
0	0.9	0

In fuzzy logica wordt voor de waarheidswaarde van p and q (= $p \wedge q$) het minimum van de afzonderlijke p en q genomen. Voor p or q (= $p \vee q$) wordt het maximum der waarheidswaarden van de p en q genomen. Daardoor is de waarheidswaarde, genoteerd met de v van valuation, van niet p: $v(\bar{p}) = 1 - v(p)$ en kunnen we $v(p \rightarrow q)$ berekenen met minimum $(1 - v(p) + v(q), 1)$.

Zadeh, de uitvinder van fuzzy sets en fuzzy logic, heeft eens gesuggereerd (Zadeh, 1972 en 1975), dat het mogelijk moet zijn de voordelen die wiskunde als één-éénduidig communicatiemiddel heeft, in vergelijking met natuurlijke taal, te gebruiken voor de constructie van een hulptaal die wel op natuurlijke taal lijkt wat betreft woordenschat, maar die toch minder dubbelzinnig is. Om die ondubbelzinnigheid te bereiken worden representaties van de te hanteren woorden in fuzzy sets gemaakt. In zijn wiskunde-betekenis grijpt dit voorstel terug naar Boole (1958) en de Poolse logicus Lukasiewicz (1967) (spreek uit: loekaazjeewiets). De laatste introduceerde meerwaardige logica's, terwijl Zadeh's representatie met behulp van fuzzy sets beschouwd kan worden als een logica met zelfs oneindig veel (waarheids-)waarden.

Woorden zoals zeer, niet, mogelijk, inderdaad, evenmin worden 'hedges' genoemd (Lakoff, 1972). Mathematisch gezien zijn het monadische operatoren op fuzzy sets. Stellen we de fuzzy set 'leeftijd' voor door een 10 steunpunten op het domain of discourse van 20 tot en met 65 jaar, dan komt er bijvoorbeeld:

AGE ← 1.0 .9 .8 .6 0 0 0 0 0 0

Nu laten we een patroonherkennend programma LABEL (voor werking zie Koppelaar, 1980) los op deze linguistische variabele AGE om de daaronder schuilende fuzzy set te herkennen en een linguistische waarde te geven. Er komt nu:

LABEL AGE → young

De 10-punts fuzzy set die toegewezen is (pijl naar links) aan AGE wordt dus geëtiketteerd (= labelled) met 'young'. In andere termen:

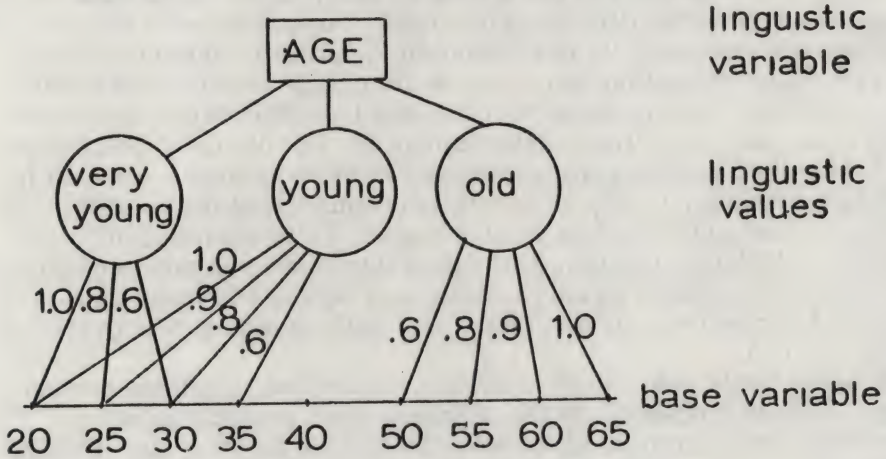
de linguistische variabele AGE heeft de waarde young. Nu laten we de hedge 'zeer' los:

VERY AGE → 1.0 .8 .6 0 0 0 0 0 0 0

Met behulp van LABEL, de patroonherkenner, wordt deze fuzzy set VERY AGE herkend als very young:

LABEL VERY AGE → very young

De hierboven gebruikte notatie voor de opslag (pijl naar links), produktie (pijl naar rechts) en herkenning (LABELing) van fuzzy sets is afkomstig uit de programmeertaal APL (= Another Programming Language) van Ken Iverson. Het programmeren van linguistische variabelen met hun labels en hedges gaat gemakkelijker in APL dan in andere talen. Om het betoog nog even samen te vatten volgt hier een tekening van de situatie, inclusief de label 'old'. Let op dat fuzzy waarden gelijk aan nul niet in het plaatje opgenomen zijn.



EXAMPLE OF LINGUISTIC VARIABLE

Eerder viel de term 'linguistic approximation'. Dit nu gebeurt in het herkenningsprogramma LABEL. Deze herkent, door een benaderingsmethode, welke label bij benadering bij een fuzzy set hoort. Voorbeelden van toepassingen van deze ideeën zijn te vinden in Koppelaar (1980) en Wenstøp (1976). De in deze paragraaf beoogde toepassing is een andere: het herkennen van indexen of indicatoren en de inhoud ervan, ook als de zoekspecificaties vaag zijn. Dat kan dus met de patroonherkenner LABEL. Een andere aanpak, die toch weer steunt op fuzzy sets, is geprobeerd door De Heer (1978). Hij werkt met zogenaamde 'informatiesporen', dat wil zeggen hij maakt van ieder

woord rijtjes van drie letters, de zogenaamde 3-strings. Voorbeeld:

WOORD wordt WOO OOR ORD

Deze informatiesporen worden gebruikt om documenten te vergelijken. De vergelijking tussen twee teksten wordt wel homeosemie genoemd. Bijvoorbeeld de ene tekst 'Organische halogeenverbindingen in drinkwater ten gevolge van chloring' wordt opgeblazen tot een informatiespoor en vervolgens gelegd tegen een, eveneens 'gespoord', databestand met 2232 samenvattingen van wetenschappelijke artikelen over watervoorziening. De artikelen zijn geschreven in diverse talen.

Een voorbeeld uit De Heer (1980), waarbij de 'passing', of 'mate van gelijkenis', de 'importance value' wordt genoemd:

EXAMPLE. A query about halogen compounds was put to the system, in English as well as in Dutch.
The English version of the query ran:

'ORGANIC HALOGEN-COMPOUNDS IN DRINKING WATER AS A CONSEQUENCE OF CHLORINATION'

The first, and so the best answer, having importance value $\zeta = 0.59$ was:

7801226 1977
MELIS, P.H.A.M.
WL 30330
75.26
Formation of organic halogen-compounds during chlorination of drinking water.
Vorming van organische halogeenverbindingen tijdens de chloring van drinkwater.
Voorburg, Rijks Inst. voor Drinkwatervoorziening, 1977, RID-Mededeling 77-1, April 35 pp., 6 figd., 7 tabs., 124 refs.
N1
Een literatuurstudie over de jaren 1972-heden betreffende de vorming van gehalogeneerde organische verbindingen tijdens de chloring van drinkwater. Een overzicht is gegeven van de verbindingen die men tot op heden als gevolg van dit proces heeft waargenomen en de verschillende reaktiemogelijkheden in waterige oplossing worden besproken. Gekonkludeerd wordt dat drie reaktie typen verantwoordelijk zijn: 1) substitutie op een N-atoom. 2) elektrofiele additie van chloor aan geaktiveerde dubbele bindingen. Bovendien is er nog vaag sprake van oxidatieve afbraak.

The Dutch version of the same query was:

'ORGANISCHE HALOGEENVERBINDINGEN IN DRINK-
WATER TEN GEVOLGE VAN CHLORING'

The same "best" answer was found as that on the English version, but now with importance value $\zeta = 0.82$, which is much higher. The reason for this may be sought in the fact that the actual description (abstract) of the content is more cognate to the Dutch version.

3.1.3 Op adem komen in LISP

1. Lists en S-expressies.

De programmeertaal LISP heeft als voornaamste eigenschap dat zij werkt met lijsten (= lists), vandaar de naam lijstverwerker (= list-processor, kortweg: LISP). Een list wordt geschreven als een rij-uitdrukking (= string-expression, kortweg: s-expression), bijvoorbeeld de letters A, B en C in een list:

(A B C)

De haakjes in S-expressions zijn altijd gepaard, bijvoorbeeld ((A B C) is geen S-expression, maar ((A B) C) wel.

Programma's in LISP worden zelf ook geschreven in S-expressions. Programma's kunnen bestaan uit primitieve functies werkend op lists, maar ook uit samengestelde functies die op lists werken.

2. Primitieve functies.

De afspraak in LISP is dat lists opgebouwd worden met behulp van zogenaamde atomen, dat zijn woorden die niet met een getal beginnen. Atomen zijn dus: Piet, totaal, lists, A, Oxy. Geen atomen zijn: 3A, 7An, 7apen.

Een andere afspraak die in LISP geldt is dat het eerste atoom na iedere openingshaak een taaknaam (functienaam) is. Dus als je LISP aanbiedt (A B C), dan vat hij A op als een functie.

Wat nu? Kent LISP dan geen data? Het tegendeel is waar. Er is een aparte functie die aan het systeem duidelijk maakt welke atomen en lijsten data zijn: de functie Quote. Deze mogen we in sommige LISP-systemen schrijven met een spoorwegkruising # en in andere met een apostroph ':

(Quote (A B C)) #(A B C) '(A B C)

betekent dat (A B C) als data, dus letterlijk (= Quoted) opgevat moet worden.

We kunnen nu andere primitieve functies bestuderen. We schrij-

ven een horizontale pijl waarvan rechts het resultaat komt van de primitieve functie.

De functie CAR pakt de eerste S-expression uit een list.

(CAR (QUOTE (A B C))) → A

Er zit ook een functie CDR in LISP die alles behalve de eerste van een list pakt.

(CDR (QUOTE (A B C))) → (B C)

De functie CONS bindt twee losse S-expressions tot een list (de punt behoort tot de folklore van LISP):

(CONS (QUOTE AAP) (QUOTE NOOT) → (AAP . NOOT)
(CONS (QUOTE A) (QUOTE (B C))) → (A B C)

De functie EQ vergelijkt twee losse atomen en geeft T als deze twee gelijk zijn:

(EQ (QUOTE X) (QUOTE X)) → T
(EQ (QUOTE X) (QUOTE Y)) → NIL

NIL is een bijzonder atoom want NIL is tevens een lege list ( ).

Er is een primitieve functie ATOM die test of een S-expression een atoom is:

(ATOM (QUOTE AAP)) → T, omdat AAP een atoom is
(ATOM (QUOTE (AAP NOOT))) → NIL, omdat (AAP NOOT) een list is

Het is mogelijk een variabele de waarde van een lijst te geven. Dit kan met de functie SETQ, bijvoorbeeld: we geven de 'onbekende' variabele X de waarde (A B C), als volgt:

(SETQ X (QUOTE (A B C))) → (A B C)

We kunnen nu testen of X een lege list vertegenwoordigt. Dit gaat met de functie NULL. X wordt deze keer niet 'QUOTED', omdat het niet gaat om X zelf (letterlijk) maar om de waarde die X vertegenwoordigt.

(NULL X) → NIL. Immers X is niet leeg.

De bekende 'als...dan...' uitdrukking heeft in LISP de volgende gedaante:

(COND ((PREDIKAAT) (DOE IETS)))

Een predikaat is een statement die iets of NIL oplevert, dus (NULL X) is er zo een. We kunnen met COND (afkorting van 'condition') bijvoorbeeld vragen om te testen of X leeg is en zo ja dan moet X de waarde (A B C) krijgen. Dit gaat als volgt:

```
(COND ((NULL X)(SETQ X (QUOTE (A B C)))))
```

In gewoon Nederlands staat hierboven: als (NULL X) waar is doe dan (SETQ X (QUOTE (A B C))). Ofwel: als X leeg is, maak X van (A B C). COND kan een willekeurig aantal argumenten hebben, bijvoorbeeld:

```
(COND ((ATOM X) X)
      (T (CAR X)))
```

Hierboven staat in gewoon Nederlands: als X een atoom is, geef dan X op, zo niet geef dan als T waar is, de eerste van X op. Het zal duidelijk zijn dat T altijd waar is, dus als X geen atoom is, wordt altijd de eerste van X opgeleverd. Let erop dat een DOE IETS regel binnen COND dan en slechts dan uitgevoerd (= geëvalueerd) wordt als het predikaat niet-NIL oplevert.

3. Het samenstellen van functies.

Met louter een pakket van primitieve functies, zoals besproken in sectie 2, is in veel gevallen een gebruiker onvoldoende toegerust voor de toepassing van LISP. Om met behulp van in LISP gedefinieerde primitieve functies meer ingewikkelde zaken te kunnen doen, gebruiken we de LISP-functie DEF om een nieuwe functie te definiëren. Bijvoorbeeld:

```
(DEF (EERSTE (X) (CAR X)))
```

In bovenstaand statement is 'eerste' de naam van de nieuwe functie en (X) is een argumentenlijst voor de nieuwe functie. (CAR X) is wat je doet doet met die argumenten. Nu kunnen we de nieuwe functie 'eerste' gebruiken.

```
(EERSTE (QUOTE (A B C))) → A
(SETQ Y (QUOTE (A B C))) → (A B C)
(EERSTE Y)               → A
```

De functie EERSTE was niet moeilijk. Nu even de functie TWEEDE. Dat gaat als volgt:

```
(DEF (TWEEDE (X) (CAR (CDR X))))
```

De laatste haakjes moeten niet vergeten worden. Immers LISP was die taal met die grote hoeveelheid gepaarde haakjes (Lots of Irritating Sexy Parentheses = L.I.S.P.). Nu kunnen we ook TWEEDE gebruiken.

```
(TWEEDE (QUOTE (A B C))) → B
(SETQ Y  (QUOTE (A B C))) → (A B C)
(TWEEDE Y)                → B
```

TWEEDE gaat de mist in als we een los atoom aanbieden. Daartoe bouwen we een COND in:

```
(DEF (TWEEDE (X) (COND
          ((ATOM X) (QUOTE ONMOGELIJK))
          (T (CAR (CDR X)))
)))
```

Let weer op de sluithaken en probeer te begrijpen wat er gebeurt:

```
(TWEEDE (QUOTE A)) → ONMOGELIJK
```

4. Verdeel en heers.

In de wiskunde wordt recursie gebruikt voor functies die vastgelegd worden in termen van zichzelf. Dit klinkt ongebruikelijk zoals wiskunde vaker klinkt en daarom nu een voorbeeld van recursieve definitie.

Het probleem dat we oplossen is de berekening van $1! = 1$, $2! = 2$, $3! = 3 * 2$, $4! = 4 * 3 * 2 * 1$, ..., $N = N * (N - 1)!$
Dit komt er voor vier faculteit: $4! = 24$. Bereken nu zelf $5!$
Als we de functie bij zijn naam noemen komt er:
Faculteit (1) = 1 en als N > 1. En N is een geheel getal dan is Faculteit (N) = N * faculteit (N - 1).

Deze functie genaamd faculteit is op deze manier recursief beschreven. Recursief omdat er de regel faculteit (N) = N * faculteit (N - 1) in voorkomt. De functie faculteit wordt in termen van zichzelf vastgelegd. Voor 'vastleggen' gebruikt men in de wiskunde meestal het woord definiëren. De definitie van faculteit in LISP luidt:

```
(DEF (FACULTEIT (N)(COND
          ((EQ N 1) 1)
          (T (* N (FACULTEIT (- N 1))))
)))
```

Dit soort recursieve definitie is toch wel erg handig. De functie

roept zichzelf aan indien niet voldaan is aan de voorwaarde dat N gelijk 1 is en er behoeft niet eens een boekhouding bijgehouden te worden voor het aflagen van N tot deze gelijk aan 1 is. Omdat er zo weinig geprogrammeerd behoeft te worden met behulp van recursieve definities heet het ook wel: het verdeel-en-heers-principe in het programmeren. Immers verdeel de te berekenen taak in tweeën: eerst het eenvoudigste geval (bijvoorbeeld faculteit(1) = 1) en vervolgens laat je de functie zichzelf aanroepen voor de moeilijkere gevallen.

Als volgende voorbeeld nemen we de functie MEMBER, die kan uitzoeken of een op te geven atoom in een te specificeren list zit. De bedoeling is dat MEMBER T geeft als het atoom daadwerkelijk in die list zit. Dat kunnen we zo definiëren:

```
(DEF (MEMBER (X Y) (COND
          ((NULL Y) NIL)
          ((EQ X (CAR Y)) T
          (T (MEMBER X (CDR Y)))
)))
```

Let in deze functie op het uitgebreide stopcriterium. Immers, voor het geval dat X in Y zit geldt dat de vergelijking van X met de desbetreffende S-expression T gaat opleveren. Echter indien die niet te vinden is, zal MEMBER Y blijven afstropen totdat het restant van Y leeg is en dan geeft het eerste deel van het stopcriterium NIL. Bijvoorbeeld:

```
(SETQ X (QUOTE (A B C))) → (A B C)
(MEMBER (QUOTE C) X)     → T
(MEMBER (QUOTE D) X)     → NIL
```

Laten we nog eens enkele functie de revue passeren! Als eerste een andere plakfunctie dan CONS, namelijk APPEND. Het verschil tussen de twee blijkt uit:

```
(CONS (QUOTE (A B)) (QUOTE C))          → ((A B) C)
(APPEND (QUOTE (A B)) (QUOTE C))        → (A B C)
(CONS (QUOTE (A B)) (QUOTE C D)))       → ((A B) (C D))
(APPEND (QUOTE (A B)) (QUOTE (C D))) → (A B C D)
```

APPEND stopt dus alle S-expressions van twee lists in één nieuwe list. Door een recursieve APPEND worden alle S-expressions van de eerste list voor de tweede geplakt.

```
(DEF (APPEND (X Y)(COND
          ((NULL X) Y)
          (T CONS (CAR X) (APPEND (CDR X) Y)))
)))
```

Ter verdere illustratie nemen we een voorbeeld van de functie EQUAL die twee lists kan vergelijken. Een uitbreiding van EQ dus. Bestudeer die:

```
(DEF (EQUAL (X Y)(COND
        ((ATOM X) (COND ((ATOM Y) (EQ X Y)) (T NIL)))
        ((EQUAL (CAR X) (CAR Y)) (EQUAL (CDR X)
                                        (CDR Y)))
        (T NIL)
)))
```

5. Het omgekeerde is hetzelfde.

In het Guinness Boek voor Records wordt melding gemaakt van de langste tekst die in omgekeerde volgorde hetzelfde gelezen wordt. Een zogenaamde palindroom. Palindromen bestaan als teksten, losse woorden en getallen. Voorbeelden:

parterretrap
9889
lepel
elisa rude lava le dur asile

Een functie die controleert of iets een palindroom is maakt gebruik van de omkeerfunctie REVERSE. In LISP is dat heel eenvoudig:

```
(DEF (REVERSE (X)(COND
        ((NULL X) NIL)
        (T (APPEND (REVERSE (CDR X)) (CONS (CAR X)
                                           NIL)))
)))
```

De functie REVERSE was een fluitje van een cent. Nu even de functie palindroom die we als predikaat programmeren, dat wil zeggen palindroom geeft T bij een palindroom en anders NIL.

```
(DEF (PALINDROOM (X) (EQUAL X (REVERSE X))))
```

Nu kunnen we de tekst 'M A D A M' checken.

```
(PALINDROOM (QUOTE (M A D A M))) → T
```

Indien een tekst niet uit losse letters of cijfers bestaat moeten we daar in voorzien, anders kan PALINDROOM niet vinden of de tekst palindroom is en zou hij dus ten onrechte besluiten tot NIL. Daar gaan we iets aan doen met een functie die alle aangeboden woorden en

getallen opblaast tot losse characters. Het gemakkelijst gaat dat met de herhaalfunctie MAPCAR. De herhaalfunctie MAPCAR pakt elk eerstvolgend (= CAR) argument en bewerkt dat volgens een door de gebruiker te specificeren functie. De functie BLAASOP levert voor ieder aangeboden los atoom een los letter-atoom met EXPLODE.

```
(EXPLODE (QUOTE MADAM) → (M A D A M)
```

De functie EXPLODE schakelen we nu herhaald in met behulp van de functie MAPCAR:

```
(DEF (BLAASOP (DATA) (MAPCAR DATA (QUOTE
                                    EXPLODE))))

(DEF (MAPCAR (X FN)(COND
          ((NULL X) NIL)
          (T CONS (FN (CAR X)) (MAPCAR (CDR X) FN)))
)))
```

De laatste definitie die in de wiskunde een functionaal heet, gaat ons misschien even boven de pet, maar hij werkt goed. Enfin, we zijn er. We kunnen nu eenvoudig testen of een tekst een palindroom vormt.

```
(SETQ TEKST (QUOTE (ELISA RUDE LAVA LE DUR ASILE)))
→ (ELISA RUDE LAVA LE DUR ASILE)
```

Bieden we tekst nu aan palindroom dan geeft hij NIL. Daarom blazen we hem woord voor woord op met onze functie BLAASOP en leggen het resultaat vast in het geheugen onder de naam TEKST2:

```
(SETQ TEKST2 (BLAASOP TEKST)) →
(E L I S A R U D E L A V A L E D U R A S I L E)
```

Het checken of TEKST2 een palindroom is gaat met:

```
(PALINDROOM TEKST2) → T,
```

met een Nederlands woord: Jaaaah!

3.2 ALS JAN MET EEN VAART VAN 3 METER PER UUR DE KAMER STOFZUIGT: METEOR

De kennismaking met programmeertalen heeft tot nu toe het volgende opgeleverd. Van de hogere programmeertalen is LISP de meest geschikte voor de tekstmachine. SNOBOL is weliswaar geschikt voor het behandelen van stukken tekst en je kunt er gemakkelijk een con-

textvrije grammatica in opschrijven, maar SNOBOL kan niet concurreren met LISP. De andere programmeertalen, talen als PL/I bijvoorbeeld, zijn alleen van belang wanneer er veel input/output moet worden gedaan. Voor taken waarbij dat zo is bestaan al veel standaardprogramma's zoals OCP. Een nadeel van LISP is echter de hoge graad van abstractie die de taal heeft. Dit wordt vooral veroorzaakt doordat de programmeur alles moet herdenken in lijsten. Wanneer nu het probleem zelf al een lijstenprobleem is (zoals bijvoorbeeld de eigenschappenlijsten bij woordvelden), is dat bezwaar niet zo groot. Er zijn echter ook andere problemen, bijvoorbeeld zinsontleding, omzetting van geschreven taal in gesproken taal, waarbij het niet zo direct voor de hand ligt om in 'lijsten' te denken. Bij dit soort problemen gaat het meer om het herkennen en omzetten van patronen. Het zou handig zijn wanneer voor de tekstmachine een programmeertaal klaar zou staan waarmee patronen konden worden herkend. Een dergelijke programmeertaal is METEOR. Deze taal werd door D. Bobrow geschreven in LISP. Hij ontwierp die taal om met de computer 'redactiesommen' te kunnen oplossen. Of liever: om in een conversatieprogramma in het Engels de computer antwoorden te laten geven op problemen die hij stelde in de vorm van redactiesommen: "Als Jan met een vaart van 3 meter per uur de kamer stofzuigt en Marie ..." Willen we onze instrumenten allemaal klaar hebben staan dan moet nu eerst de kennismaking met METEOR komen.

METEOR bestaat uit regels. Deze regels hebben een vaste vorm die beschreven wordt als rechterkant en linkerkant:

Algemene vorm van een METEOR REGEL:

(* *)

Moet een regel herhaald worden dan schrijft men in plaats van de asterisk voor de sluithaak een slash (/). Een regel die herhaald moet worden ziet er dus als volgt uit:

(* /)

De helften binnen de regel worden omsloten door ronde haakjes (we zijn immers bezig in LISP!). Daarbij bevat de linkerhelft het patroon dat moet worden gezocht in de tekst en de rechterhelft de bewerkingen die moeten worden verricht wanneer het patroon van links werd gevonden.

(*	( )	( )	*)
opening	linkerhelft	rechterhelft	sluiten

3.2.1 Werken met METEOR

In voorbeelden geven we de mogelijke operaties met METEOR regels aan.

3.2.1.1 Vervanging (replacement)

(* (PIET) (KEES) *)

Deze regel geeft een enkelvoudige vervanging. Het woord PIET wordt vervangen door het woord KEES.

(* (IK VOEL) (VOEL JIJ) *)

Dit zou dus een van de operaties zijn die een ELIZA-programma in METEOR zou kunnen gebruiken.

Deze regel zou men ook algemener of abstracter kunnen noteren in METEOR. Als volgt:

(* (IK VOEL) (2 JIJ) *)

Uit deze regel kan men het volgende opmaken. Het aantal elementen of constituenten, genoemd in de linkerhelft, kan men aangeven met getallen, in die zin dat het eerste element correspondeert met het getal 1, het tweede met het getal 2, enz. Hierdoor ontstaat een snellere manier van noteren en hoeft men slechts dat wat werkelijk verandert op te geven.

- Toepassing

Met deze zeer eenvoudige regels kan men nu zelf al problemen programmeren waaraan in de beginjaren van de computerlinguistiek officiële 'wetenschappelijke' artikelen werden gewijd. Een van die problemen is het omzetten van zinnen uit de zogenaamde directe rede in de indirecte rede.

Voorbeeldzin:

Jullie zeggen: "Het is mooi weer."

In de indirecte rede luidt deze zin:

Jullie zeggen dat het mooi weer is.

Om van de directe rede naar de indirecte rede te komen moet dus het volgende gebeuren:

1. de leestekens (dubbelpunt en aanhalingstekens) moeten verdwijnen respectievelijk vervangen worden door het woordje 'dat';
2. het woordje 'is' moet verhuizen naar de laatste plaats in de zin.

In METEOR aldus:

```
(*    ( : " )          ( DAT )    /)
(*    ( IS $ ")        ( 2 1 )    /)
```

In het voorbeeld hebben we een zeer machtige operator in de linkerhelft gezien, namelijk de $ (de DOLLAR MATCH). Deze operator betekent: neem alles wat er staat. In ons geval: neem alles wat er staat tussen 'is' en het eerstvolgende aanhalingsteken en plaats dat voorop, daarna komt het woord 'is'.

OPGAVE.
Schrijf nu de METEOR regels die de volgende zinnen omzetten van directe rede naar indirecte rede.

Hij zegt: "het sneeuwt buiten"
Zij zegt: "ik ben zeker gek, hè?"
Ik denk: "wát is het toch prettig om meteor te leren."
De algol freak zegt: "die taal is niet netjes."
De fatsoensrakker denkt: "ik moet algol gebruiken."

- Nog een paar problemen bij de indirecte rede

Het ruwe canvas voor de oplossing van het probleem der indirecte rede is bevestigd. Het zou echter niet om taal en tekst gaan wanneer nu al het hele probleem zou zijn opgelost. Kijk maar eens naar de volgende zinnen:

> Hij vraagt: "Is het probleem nu echt opgelost?"
> Ze vroeg ook: "Is het probleem nu echt opgelost?"

Wanneer men deze twee zinnen van de directe rede omzet naar de twee correcte Nederlandse zinnen in de indirecte rede blijken twee dingen. De zinnen luiden:

> Hij vraagt <u>of</u> het probleem nu echt opgelost is.
> Ze vroeg ook of het probleem nu echt opgelost <u>was</u>.

De problemen luiden als volgt:

> Na vragen komt niet <u>dat</u> maar <u>of</u>.
> Staat de hoofdzin, d.w.z. de zin buiten de aanhalingstekens, in de verleden tijd, dan moet ook de om te zetten zin in de verleden tijd.

Het eerste probleem is eenvoudig op te lossen. Namelijk op dezelfde manier waarop men 'dat' inbracht, kan 'of' worden geregeld.

• Wisseling der tijden

Het tweede probleem is ingewikkelder, want daarvoor moet men de verleden tijden van een werkwoord kunnen herkennen.

Daarvoor kan men twee dingen doen. Men kan alle verleden tijden opsommen. Dit is een soort woordenboekoplossing. Of men kan een programma schrijven dat de verleden tijd zelf herkent. Het is duidelijk dat de tweede oplossing intelligenter is dan de eerste. De eerste oplossing (de opsom-methode) werd in het verleden gebruikt. In de tekst werd dan met de hand ingecodeerd waar wat voor soort werkwoord in de tekst voorkwam. Zoals het spreekwoord zegt: 'Wie niet denkt, moet zijn benen gebruiken.' Een tekstmachine zou natuurlijk de intelligentere oplossing moeten kiezen, want wat is die tekstmachine nog waard, wanneer al bij het eerste het beste probleem de mens handwerk gaat zitten verrichten?!

Overigens zijn er nog meer problemen bij het omzetten van directe naar indirecte rede:

Hij zei: "Je bent niet goed snik."
wordt: Hij zei, dat ik niet goed snik was.

Hij vroeg aan zijn broer: "Je lust er zeker nog wel eentje."
wordt: Hij vroeg aan zijn broer of hij er nog een lustte.

Toch zijn dit kleine probleempjes wanneer men eenmaal het probleem van de wisseling der tijden heeft opgelost.

In de loop van dit hoofdstuk zullen we met technieken en suggesties komen ter oplossing van dit probleem. Nu echter zou het ons teveel afleiden van het doel van deze paragraaf: de werking van METEOR. Het blijkt echter nu al dat een hogere programmeertaal als METEOR direct de eigenlijke problemen serveert. Men kan zich nu niet meer verschuilen achter de schijngestalten van de ALGOL-maan. Hier blijkt dan ook het belang van een echt hoogontwikkelde programmeertaal voor text processing.

3.2.1.2 Weglaten (deletion)

Regel

```
(*    (KEES)              (0)    /)
(*    (KEES EN MARIE)     (0)    /)
```

Dat wil zeggen:
regel 1 betekent: alle keren dat in een tekst het woord KEES voorkomt wordt dit woord geschrapt;
regel 2 betekent: alle keren dat de combinatie KEES EN MARIE voorkomt in een tekst wordt deze combinatie geschrapt.

• Toepassing

Men zou zich weglaten als operatie kunnen voorstellen bij het proces van zogenaamde lemmatisering. Lemmatisering wil zeggen: men herleidt een verbogen woordvorm tot een gekozen andere vorm, bijvoorbeeld tot de zogenaamde stam. De stam van het woord 'werken' is WERK. Wil men lemmatiseren tot op de stam dan moeten alle woordvormen die kunnen worden gemaakt met de stam WERK daartoe worden herleid. Dus:

werk
werkt
werkte } WERK
gewerkt
werken

Stel dat men alleen deze woorden wil herleiden en niet bijvoorbeeld ook werktafel en achterwerk, dan zou een serie regels waarin gebruik gemaakt wordt van de weglaatoperator er als volgt uitzien:

```
(*    ( T )      ( 0 )      /)
(*    ( GE )     ( 0 )      /)
(*    ( TE )     ( 0 )      /)
(*    ( TEN )    ( 0 )      /)
(*    ( EN )     ( 0 )      /)
```

• Een kleine aanvulling: de opblaas- of expand-operator

Het idee van het bovenstaande programma is juist. Toch zou het in deze vorm niet werken. Immers de elementen uit een tekst die worden herkend zijn alle elementen die worden omsloten door spaties. In 'werkt' wordt alleen het hele woord herkend en niet de T alleen, waarom in regel 1 wordt gevraagd. Daarom moet er eerst voor gezorgd worden dat de tekst bestaat uit losse letters. In METEOR gebeurt dat met de expansie-operator waarvan de vorm de volgende is:

Regel

```
(*    ( ($.1) )        ( (*E 1 ) )    *)
```

Dat wil zeggen: zoek een element dat bestaat uit een ATOM (de DOLLAR MATCH voor 1 element heet $.1) en voer daarop de expansie-operator uit (zie de opdracht van de rechterhelft).

Het is duidelijk dat nu ook in de regels van het eerste programma niet meer letterpatroontjes als TE mogen voorkomen, maar dat die nu moeten worden geschreven als T E.

OPGAVE. Schrijf de METEOR regels die nodig zijn om het woord ANTWOORDEN te lemmatiseren.

• Samenvatting

We hebben nu de ruwe vorm van het programma geschreven dat nodig is om alle regelmatige zwakke werkwoorden in het Nederlands te lemmatiseren. We hebben daarbij ook een gedeelte van de zelfstandige naamwoorden gevangen. Het complete programma bestaat uit een tiental regels.

3.2.1.3 Omordening (rearrangement)

De omordening werd behandeld in het voorbeeld over de omzetting van directe rede in indirecte rede (p. 115).

• Combinatie van weglating en omordening

Voorbeeld:

```
(*    ( m e n u u t j e )    ( 1 2 3 4 (* DASH ) 6 7 8 )    *)
```

De voorbeeldregel levert als output op de afbreking van het woord 'menuutje' (verkleinwoord van menu) en wordt aldus geschreven: menu-tje. Uit de constituenten in de linkerhelft is het vijfde element weggelaten. Dat kan worden opgemaakt uit het feit dat in de rechterhelft geen verwijzing '5' voorkomt. In plaats daarvan is een afbreekstreepje ('hyphen') opgenomen; in METEOR is dat de (* DASH).

• Speciale tekens (special characters)

In het voorbeeld 'menuutje' werd als afbreekteken niet eenvoudigweg gebruikt het teken '-'. Dit teken moest voor METEOR worden omschreven. Dit komt doordat er een paar tekens gereserveerd zijn voor LISP-operaties. Het gaat daarbij om de volgende tekens (met hun METEOR-naam):

TEKEN	METEOR-naam
.	(* PERIOD)
,	(* COMMA)
+	(* PLUS)
(	(* LPAR)
)	(* RPAR)
-	(* DASH)

3.2.1.4 Invoegen (insertion)

Regel:

```
(*    ( EEN RODE BLOEM )    ( 1 2 3 IS EEN ROOS )    *)
```

levert op: EEN RODE BLOEM IS EEN ROOS.

• Toepassing: Verkleinwoorden

Verkleinwoorden moeten dikwijls worden uitgebreid, hetzij met 'je', hetzij met 'tje', hetzij met nog meer dan dat.
Voorbeelden:

ski - skietje
opa - opaatje
menu - menuutje
gebed - gebedje
meester - meestertje
blauw - blauwtje
heg - hegje

De METEOR regels voor deze woorden en alle gelijke gevallen zijn:

```
(*    ( I )        ( 1 E T J E )    /)
(*    ( A )        ( 1 A T J E )    /)
(*    ( U )        ( 1 U T J E )    /)
(*    ( D )        ( 1 J E )        /)
(*    ( R )        ( 1 T J E)       /)
(*    ( W )        ( 1 T J E)       /)
(*    ( G )        ( 1 J E )        /)
```

Het is duidelijk dat men zo kan doorgaan tot alle regels voor de verkleinwoorden zijn opgeschreven. Wat opvalt is nogmaals dat men zijn regels definieert op het niveau van het probleem zelf en dat er geen computer hokus pokus aan te pas komt.

• Een addertje onder het gras: het eind van het woord

Neemt men de regels zoals ze boven beschreven werden op in een programma dan zal het zeker mis gaan, want alle voorkomende letters 'a' worden in dat programma vervangen door 'aatje'. Dus mamma wordt dan maatjemmaatje. Dat is niet de bedoeling, al krijgt men wel verrassende en leuke woorden. De plaats waar de toevoeging moet worden gedaan is het eind van het woord. Dat is dus de plaats waar rechts van de letter niets meer te lezen is. De controle of er ergens niets meer te lezen is geschiedt met de DOLLAR-0 MATCH: ($. 0).
De eerste regel van het programma (en alle andere analoog) moet dus als volgt luiden:

```
(*    ( I ($. 0) )        ( 1 E T J E )    /)
```

OPGAVEN

1. Herschrijf het verkleinwoordenprogramma in zijn correcte vorm.
2. Schrijf de METEOR regels voor de verkleining van de volgende woorden: ster, café, bloem, man, hof, ei, geel, hiel, koning.

3. Schrijf het varkenslatijn in METEOR.
4. Schrijf in METEOR een woord-voor-woord vertaler met 10 Engels-Nederlandse woorden.

3.2.1.5 Doorgeven van tussenresultaten (debugging)

Het kan voorkomen dat een programma resultaten oplevert die niet waren voorzien en die niet waren bedoeld. Dan moet men in staat zijn na te gaan waar de onverwachte resultaten vandaan komen. Er zijn een serie mogelijkheden om de gang van zaken binnen het programma te controleren.

De eerste en meest voor de hand liggende is de logica van het programma zelf controleren, ofwel: beter lezen. Nu kan het echter zijn dat beter lezen de blindheid niet doet verdwijnen. Men zou dan wel tussenresultaten willen afdrukken. Daarvoor zijn drie operatoren beschikbaar: de *P, de *M en de *T.

- De *P operator

Met de *P operator kan men de stand van zaken afdrukken zoals die zich op een gegeven moment bevindt in het werkgeheugen (de WORK-SPACE).
Voorbeeld:

```
(*      (OLIFANT)          ( (*E 1 ) )      *)
(*      ( (*P  ) )                          *)
```

levert op: (O L I F A N T).

Schrijft men in plaats van ((*P)) nu ((*P DE TOESTAND IS)) *), dan levert dat op:

```
( DE TOESTAND IS )
( O L I F A N T )
```

- Tracen (de *T operator en de *U operator)

Het kan zijn dat men met geen goede wil meer kan achterhalen waar men een fout moet zoeken. Dan moet men een outprint hebben van alles wat het programma doet. Ofwel dan moet men alle toestanden weten zowel van voor de aanroep van een regel als erna. Dit heet tracen en het gebeurt met de *T operator als volgt:

```
(*   *T   ( R )          ( 1 R E T J E )     /)
(*        ( M )          ( 1 P J E )         /)
(*        ( L )          ( 1 T J E )         /)
```

Mocht dit een programma of een onderdeel van een programma zijn, dan zou dat programma alles wat er gebeurt afdrukken. De *T moet dan wel op de tweede plaats in een regel staan, dat wil zeggen voor

de linkerhelft. Te beginnen met de regel waar de *T staat wordt alles afgedrukt wat het programma doet.

- De *U operator

Het kan zijn dat men wel weet in welk gedeelte van het programma de fout zit, maar niet in welke regel precies. Wanneer men weet tot waar in het programma getraced moet worden kan men bij de eerste regel daarna de tracing weer uitschakelen. Dat gebeurt met de *U operator die op dezelfde plaats, dat wil zeggen in de tweede positie in de regel moet voorkomen. Aldus:

```
(*  *T   ( R )          ( 1 R E T J E )     /)
(*       ( M )          ..............
..............
..............
..............
..............
(*  *U   ( L )          ( 1 T J E )         /)
```

- De *M operator

Stel dat men weet dat er iets mis is met een enkele regel, maar dat men niet begrijpt wat. Ook in dat geval zou men partieel willen tracen, maar dan natuurlijk wel alleen maar de regel in kwestie. Dat gebeurt met de *M operator. Als volgt:

```
(*      ( ..... )          ( ..... )          /)
          .....              .....
          .....              .....
(*  *M    ( R )            ( 1 R E T J E )    /)
          .....              .....
          .....              .....
          .....              .....
```

3.2.1.6 Het geven van commentaar in de programmatekst zelf (COMMENT)

Het is mogelijk dat men zijn programmateksten leesbaar wil houden door aan te geven waar een regel of een serie regels voor bedoeld is. Het commentaar dat men wil meegeven wordt tussen ronde haken geschreven rechts van de rechterhelft van een regel, vóór de asterisk met een spatie rechts en links ervan. Aldus:

```
(*      ( R )      ( 1 R E T J E ) * ( voor gevallen als
                                       sterretje )      /)
```

3.2.1.7 Het overslaan van een serie regels: GO TO sectie

Het komt veel voor in computerprogramma's dat men een serie instructies moet overslaan omdat ze in strijd zijn met net gevonden toestanden en/of gegevens. De normale programmeertalen gebruiken in dit soort gevallen een sprongopdracht die meestal bestaat uit de tekst GO TO (plus een zogenaamde label waarheen gesprongen moet worden). In METEOR bestaat de mogelijkheid van springen ook. Als volgt:

```
(*      (       )          (       )      AA)
```

De letters AA voor de sluithaak geven aan waarheen gesprongen moet worden. Voorbeeld: Stel men is bezig een METEOR programma te schrijven voor woordafbreking. Een van de problemen daarbij wordt gevormd door de lettercombinatie AF. Dit kan in het Nederlands namelijk een voorvoegsel zijn, maar ook niet. Zie bijvoorbeeld de woorden:

afrekenen
afbekken
afrika
afroaziatisch

Men zou dus graag een regel willen schrijven om het voorvoegsel 'af' te vangen. Ook echter zou men, wanneer men de uitzonderingen zoals 'afrika' en 'afroaziatisch' heeft gevonden, niet nogmaals willen gaan controleren op het voorvoegsel 'af'. Dat kost immers onnodig rekentijd. Welnu, in een dergelijk geval gebruikt men een sprong (jump). De METEOR regels voor ons voorbeeld zien er als volgt uit:

```
(*      ( A F R I )      ( 1 (* DASH ) 2 3 4 )      AA)
(*      ( A F R O )      ( 1 (* DASH ) 2 3 4 )      AA)
(*      (  ($.0) A F ($.1)  )    ( 2 3 (  DASH ) 4 )      *)
(AA     (       )          (       )          *)
```

VRAGEN

1. Is het noodzakelijk om in de derde METEOR regel de DOLLAR MATCH ($.0) te gebruiken?
2. Is het noodzakelijk om in de derde regel de DOLLAR MATCH ($.1) te gebruiken?
3. Hoe ziet de output eruit wanneer de bovenstaande drie METEOR regels worden toegepast op de vier voorbeeldwoorden (afrekenen, afbekken, afrika, afroaziatisch)?

3.2.1.8 Property-list in METEOR: subscripting

In het eerste deel van dit hoofdstuk was uitgebreid sprake van lijsten van eigenschappen die men onder een bepaald hoofd (een woord of een 'label') zou willen samenvatten. We noemden dit met de vakterm: property lists. In METEOR kan men ook woorden als label gebruiken. Men kan er dus informatie over meegeven. Dat gebeurt aldus:

```
(VIS / SCHOL HARING DOLFIJN ( LEEFT IN HET WATER)
       KOUDBLOEDIG )
```

Wil men de aldus opgeslagen informatie gebruiken dan kan dat op verschillende manieren. Ten eerste kan men zoeken naar VIS. In dat geval vindt METEOR het geheel. Ofwel: bij zoeken moet het atoom links van de / die de eigenschappen aangeeft worden opgegeven. Aldus:

```
(*      ( VIS )          (      )      *)
```

Nu kan het echter ook zijn dat men alle koudbloedige dieren wil vinden. Dat wil zeggen dat men wil zoeken vanuit de eigenschappenlijst en niet vanuit de labels. Men kan in dit geval niet zoeken vanuit de label, omdat men juist zo graag zou willen weten wat die labels zijn. Men moet dus een DOLLAR MATCH gebruiken. De vraag is nu namelijk: zoek alle atomen waarachter in de property list koudbloedig staat. De regel moet aldus luiden:

```
(*      ( ( $.1)    / KOUDBLOEDIG )      (      )      /)
```

Opmerking: In tegenstelling tot de BNF-notatie of tot LISP doet het in METEOR niet ter zake hoe de properties geordend zijn. KOUDBLOEDIG had dus net zo goed voorop kunnen staan of middenin.

3.2.1.9 Toevoegen van eigenschappen

Data-bestanden die men wil gebruiken ondergaan regelmatig veranderingen. Dat bleek ook al bij de opbouw van het databestand DIEREN (pp. 98 e.v.), waarmee we hebben gedemonstreerd hoe een intelligente data-base moest worden opgebouwd en gebruikt. Telkens moesten eigenschappen worden toegevoegd of eventueel weggelaten. In METEOR zijn ook hiervoor functies aanwezig en wel de atomen AND, OR en SUBST. Daarbij staat AND voor toevoegen, OR voor kiezen en SUBST voor vervangen.

Voorbeelden.

```
a. (*   (     )        (*/ OR JONGEN ( MAN / MANNELIJK
                  ZELFSTANDIG NAAMWOORD ) )     *)
```

Bij toepassing van deze regel is het resultaat, tenminste wanneer JONGEN werd gevonden in het bestand:

```
(JONGEN  /  MANNELIJK ZELFSTANDIG NAAMWOORD )
```

```
b. (*   ( ( JONGEN / NAAMWOORD ENKELVOUD ) )
                ( (* AND 1 (HOND / NAAMWOORD
                MANNELIJK ) )    *)
```

Deze regel levert op:

```
(JONGEN / NAAMWOORD )
```

```
c. (*   ( (JONGEN / NAAMWOORD ENKELVOUD ) )
                ( (* / OR 1 ( JONGEN / KLEIN
                MANNELIJK ) )    *)
```

Deze regel levert op:

```
(JONGEN / NAAMWOORD
    ENKELVOUD KLEIN MANNELIJK )
```

```
d. (*   ( ( JONGEN / NAAMWOORD ENKELVOUD ) )
            ( (* / SUBST 1 (AAP / MANNELIJK ) )    *)
```

Deze regel levert op:

```
(JONGEN  / MANNELIJK )
```

3.2.1.10 Zolang iets ergens wegzetten (temporary storage)

Het kan zijn dat men iets gevonden heeft in een tekst dat men zolang wil bewaren. Bijvoorbeeld: Men heeft bij zinsontleding het onderwerp gevonden. Voor de rest van het onderzoek zou men graag weten welke woorden voorkomen in het onderwerp van de zin. Dan kan men ieder gevonden onderwerp wegschrijven naar een tijdelijke plaats. Deze plaats wordt genoemd een SHELF. Deze SHELVES kan men bedienen in de rechterhelft van een regel. Dit gedeelte van de regel wordt genoemd de verzendafdeling (ROUTING SECTION). De verzendafdeling is een lijst in de rechterhelft die begint met een slash, als volgt:

```
(/ (....) )
```

- Operatoren voor de ROUTING SECTION

De volgende operatoren kunnen worden gebruikt in de verzendafdeling:

```
*A      *N      *P      *Q      *S
```

- De *S operator

Met de *S operator wordt datgene wat op een tijdelijke plaats wordt weggeschreven voorop op de shelf geplaatst. Haalt men de elementen weer tevoorschijn (dat gebeurt met de *N operator), dan komt eerst het laatst bijgeplaatste element tevoorschijn. In informaticatermen noemt men een shelf die volgens deze specificaties werkt een PUSH DOWN LIST. Een push down list werkt als een stapel pannekoeken: men eet de pannekoek die het laatst gebakken is het eerst op, want die ligt bovenop de stapel. Ook noemt men dit wel LIFO, de afkorting voor Last In First Out.

Voorbeelden.

a. Wegzetten

```
(*    ($)    (/   (*S   WEG 1)  )    *)
```

Met deze regel wordt een willekeurig gevonden element (DOLLAR MATCH) weggezet op de push down list die de naam krijgt WEG.

b. Ophalen

```
(*    ($)    (    (*N   WEG)   )    *)
```

Met deze regel wordt het laatste element op de shelf WEG teruggeplaatst in de WORKSPACE.

- De *Q operator

Voor de *Q operator geldt hetzelfde als voor de *S operator, zij het dat men nu een lijst heeft gemaakt van het type FIFO, dat wil zeggen First In First Out, ofwel het element dat het eerst op de lijst werd geplaatst komt er ook het eerst uit. Deze operator is handig wanneer men een tekst woord voor woord wil gaan behandelen, terwijl men toch de woorden in hun oorspronkelijke volgorde moet laten. Te denken valt bijvoorbeeld aan een programma voor woordafbreking. De eerste METEOR regel daarvan zal luiden:

```
(*    ($)    (/   (*Q   WEG   1)  )    *)
```

Ophalen (woord voor woord) gebeurt met de volgende opdracht:

```
(*    ($)    (    (*N   WEG)   )    *)
```

- De *X operator

Een wegzendinstructie die niet begint met *S of *Q maar met *X vervangt de hele inhoud van een SHELF door wat er in de WORKSPACE, dat is het werkgeheugen, staat.

• De *P operator

Begint een wegzendinstructie (routing instruction) met *P waarna de naam van een SHELF genoemd wordt, dan drukt de machine af:

SHELF / naam van de SHELF / CONTAINS / inhoud van de SHELF

Voorbeeld:

```
(*    ( )    ( )    (/   (*P ONDERWERP ) )    *)
```

geeft de output:

```
SHELF onderwerp CONTAINS      / met hier alles wat er op die
                                SHELF staat /
```

Wil men de inhoud van alle SHELVES afgeprint zien, dan schrijft men

```
(/    (*P   / )  )
```

• De *A operator

Gebruikt men de *A operator, dan plaatst de machine alles uit een te noemen SHELF in de WORKSPACE.

• De *N operator

Gebruikt men de *N operator dan plaatst de machine het volgende element in de WORKSPACE. Daarbij is het van belang te weten van wat voor soort SHELF de informatie moet komen: van een FIFO of van een LIFO.

3.2.1.11 Compressie

Soms is het wenselijk verschillende elementen in de WORKSPACE samen te voegen tot een nieuw element. Bijvoorbeeld wanneer men zinnen redekundig ontleedt, verdient het aanbeveling de herschrijving van de nominale groep tot NP samen te vatten in een enkele lijst omgeven door haakjes. Als volgt:

```
(NP  ((de / det )  (man  / subst ) ) )
```

Deze operatie wordt compressie genoemd. Compressie geschiedt in de rechterhelft van een METEOR regel en door gebruik te maken van de *K operator.

• De *K operator

De algemene gedaante van een METEOR regel met de *K operator is als volgt:

```
(*    ( $ )    (*K 1)    *)
```

Voorbeeld: Men wil bij zinsontleding de woordgroep die bestaat uit 'de man' samenvatten tot NP. Dat gaat als volgt:

```
(*    ( DE MAN )    (    (*K    NP / 1 2 )    *)
```

OPGAVE. Schrijf de METEOR regel waarin a: alle combinaties van 'de vrouw' in een willekeurige tekst worden herschreven tot NP en waarin tegelijkertijd b: alle woorden die tussen 'de' en 'vrouw' staan worden meegenomen.

- De *C operator

Wil men losse letters compressen dan gebruikt men niet de *K operator maar de *C operator, die precies zo dient te worden gebruikt als de *K operator. Men krijgt natuurlijk wel een foutmelding wanneer de elementen die met de *C worden gecomprimeerd niet uit losse letters bestaan.

3.2.1.12 Expansie: de ⋆E operator

Wanneer men woorden inleest en men wil op letterniveau gaan opereren, dan moet men de woorden eerst omzetten in losse letters. Dat gebeurt met de *E operator. De *E operator kan men ook gebruiken om compressies weer ongedaan te maken.

Voorbeeld. Men leest eerst een hele tekst in. De afzonderlijke woorden worden op een FIFO SHELF geplaatst. Eén voor één worden de woorden van de SHELF gehaald en geëxpandeerd tot reeksen losse letters, gescheiden door een 'blanc'.

```
(*    ( $ )      (/    (*Q   WEG ) )    *)
(*    ( $ )      (/    (*N   WEG ) )    *)
(*    ( ( $.1)   (     (*E   1   ) )    *)
```

3.2.1.13 Lezen en schrijven: de ⋆R en de ⋆W operator

In de rechterhelft van een METEOR regel kan men woorden wegschrijven met de *W operator. Als volgt:

```
(*    ($)    (*W/ naam1 naam2 naam3 ..... naamk )  *)
```

Let erop dat losse symbolen (zoals komma en punt) op een speciale manier moeten worden aangeduid!

Lezen gaat met de *R operator als volgt:

```
(*    ( $ )    ( (*R) )    *)
```

Met deze regel leest de machine van ponskaart of van een file alle characters die op een record te vinden zijn. Deze informatie plaatst hij in de WORKSPACE.

Voorbeeld:

```
(*   ( $ )      ( 1 (*W IK GA NU PROBEREN IDIOMATISCHE
                    UITDRUKKINGEN TE VINDEN  $EOR$ )  )    *)
```

Opmerking: Het atoom EOR duidt aan dat de computer na deze zin aan het eind van een record is gekomen en dus op een nieuwe regel zal vervolgen.

```
(*      ( $ )      (  (*R )  )      *)
```

Deze regel vervangt alles wat er in de WORKSPACE staat door de informatie die gevonden wordt op een record van een input medium (bijvoorbeeld een terminal).

3.2.2 Samenvatting en overzicht van METEOR (formeel)

Vanwege het feit dat METEOR geschreven is in LISP is ieder statement een list bestaande uit sublists en losse atomen. De vrijwel algemene gedaante van ieder METEOR statement is:

(name1 transformationrule routingsection name2 comment)

Het atoom 'name1' werkt als label en mag dus iedere naam zijn, mits deze begint met een letter. Indien er geen sprong naar 'name1' gedaan wordt, is het overbodig de statement van een label te voorzien en kan 'name1' vervangen worden door '*'.

De transformation rule bestaat uit twee elementen achter elkaar, waarvan de eerste een list moet zijn en de tweede optioneel is. Het eerste element, de list, wordt 'left-half-pattern' genoemd en deze bevat de zoekopdrachten met betrekking tot de data. Het tweede element van de transformation rule, het zogenaamde 'right-half-pattern', mag leeg zijn; dat betekent dat, indien er een 'matching' (d.i. de zoekopdracht is gelukt) plaatsvindt, er in de dataset niets verandert. Is evenwel het 'right-half-pattern' wel expliciet gedefinieerd, dan wordt de dataset zo veranderd als in het 'right-half-pattern' is beschreven.

Het kan nodig zijn tijdelijk een gedeelte van de data uit de WORKSPACE te halen zonder dat die informatie verloren gaat. Dit gebeurt door middel van de routing section. In deze 'routing section' wordt bepaald hoe en waar de informatie wordt opgeslagen. Even-

als met het 'right-half-pattern' is het zo dat de 'routing section' uitgevoerd wordt als er een 'matching' plaatsvindt (de zoekopdracht in de 'left-half-pattern' slaagt).

In het algemeen kan men zeggen dat wat NA de 'left-half-pattern' staat alleen uitgevoerd wordt als de zoekprocedure succes heeft.

'Name2' is de naam van de METEOR regel waar het programma naar toe moet springen nadat het voorgaande gedeelte van de statement is uitgevoerd. Na dit globale overzicht hoe een METEOR regel eruit ziet en hoe deze verwerkt wordt, zullen we de elementen van een regel uitgebreid behandelen.

- Basisgereedschap in METEOR

Het belangrijkste van een METEOR-programma zijn de zoekprocedures. Het is daarom noodzakelijk dat de procedures zo duidelijk mogelijk gedefinieerd worden, en tevens dat ze zo efficiënt mogelijk zijn. Dit houdt o. a. in dat twee of meer METEOR regels zo mogelijk samengebald worden tot één.

Om dit goed te kunnen is het belangrijk te weten wat de mogelijkheden binnen de zoekprocedures zijn. Deze zoekprocedures worden beschreven in het 'left-half-pattern', en dat is een lijst bestaande uit minstens één van de volgende elementen:

$, ($. n), ($. atom), ($. list), ($. number), number, atom, list.

$: Het dollarteken veroorzaakt altijd een matching met de datatest (= workspace). Een ongedefinieerd aantal elementen in de workspace wordt gevonden (dus 0 wanneer de workspace leeg is!). Voor de matching is de gedaante van de data-elementen niet van belang. Dit kunnen zowel atomen zijn als lijsten en getallen.

($. n) : Hiermee wordt het aantal elementen dat gezocht wordt in de workspace wèl gedefinieerd.
Is $n \geq 1$ dan worden n opeenvolgende elementen in de workspace gezocht.
Is $n = 0$ dan wordt òf het begin òf het eind van de workspace bedoeld. Dit is afhankelijk van de plaats van ($. 0) in de 'left-half-pattern'.

($. atom) : Door dit in de zoekprocedure te plaatsen wordt gezocht naar het eerst voorkomende atoom in de workspace.

($. list) : In plaats van een atoom bij ($. atom) wordt nu gezocht naar de eerste lijst in de workspace.

($. number) : Nu wordt gezocht naar het eerste getal.

number : Number staat voor een willekeurig getal. Bij de zoekprocedure wordt eerst gekeken of dit getal een label is van een (in dezelfde 'left-half') gevonden element. Is dit het geval dan wordt gezocht naar dit element, anders naar het getal zelf.

atom : Dit staat voor een willekeurig atoom.

list : List staat voor een willekeurige lijst.

Als de zoekactie gelukt is gaat de verwerking van de 'right-half-pattern' beginnen. In de 'right-half-pattern' staat de transformatie die uitgevoerd moet worden op de workspace. Dit kan op een aantal manieren gebeuren:

a. reorganisatie;
b. compressie en expansie;
c. gedeeltelijke 'shelf-handling'.

• Reorganisatie

Reorganisatie betekent, behalve een verandering van de rangschikking van de elementen, ook duplicatie, reductie en het toevoegen van elementen aan de workspace.

Verandering van de rangorde van de data-elementen gebeurt door de rangschikking van de 'left-half' labels dienovereenkomstig te wijzigen in de 'right-half-pattern'. Wil men één of meer elementen toevoegen in de workspace, dan is het voldoende deze elementen in de gewenste volgorde en met het gewenste aantal herhalingen in de 'right-half' te noemen.

• Compressie en expansie

Compressie en expansie kan op twee niveaus toegepast worden, namelijk op lijsten en op de elementen van lijsten.

Expansie van een lijst houdt in dat die lijst vervangen wordt door de elementen van deze lijst, terwijl bij expansie van een atoom dit atoom wordt opgesplitst in de losse karakters waaruit dit atoom bestaat. De aanduiding voor 'expandeer x' in de 'right-half-pattern' luidt: (*E x). Hierbij staat 'x' voor een of meer labels van elementen in de workspace of een atoom of een lijst.

Er zijn twee operatoren om compressie te bewerkstelligen. De 'right-half-pattern' element (*K x), waarbij de x staat voor een of meer elementen uit een lijst.

Begint dit 'right-half-pattern' element echter met '*C', dan worden de daarna genoemde 'workspace'-elementen gecomprimeerd tot een atoom. Voorwaarde hiervoor is wel, dat de elementen losse karakters zijn. Is dit niet het geval dan treedt een foutmelding op en stopt het programma.

• Gedeeltelijke 'shelf-handling'

Gedeeltelijke 'shelf-handling' maakt het mogelijk het eerste element of alle elementen van een 'shelf' (zie verderop) te halen. Hiervoor moet men in de 'right-half-pattern' het element (*N naam1) of (*A naam1) plaatsen. Naam1 staat voor de naam van de bedoelde 'shelf'.

• 'Shelves'

Het is regelmatig nodig gedeelten uit de 'workspace' tijdelijk in het geheugen van de computer op te slaan. Binnen METEOR zijn hiervoor de zogenaamde 'shelves' gemaakt. Deze 'shelves' hebben een bijzonder grote capaciteit, maar hun aantal is beperkt tot 127.

Een ander punt is dat het gebruik van 'shelves' relatief gezien veel tijd kost. Het verdient dus aanbeveling zo min mogelijk 'shelves' te gebruiken en na te gaan of een 'shelf' echt wel nodig is voor het te programmeren probleem.

Het creëren van 'shelves' gebeurt in de 'routing section'. Deze 'routing section' is een lijst van de volgende structuur: (/ lijst1 ... lijstn). Hierin geeft '/' aan dat erna een 'shelf-handling' plaatsvindt. 'Lijst1' staat voor (op1 naam1 x). Dit betekent: aan de 'shelf' met de naam 'naam1' worden op een door 'op1' te bepalen wijze de elementen toegevoegd die door x worden beschreven. Het creëren van 'shelves' wordt gestuurd door drie operatoren, namelijk *Q, *S en *X.

*Q : Deze operator creëert een 'shelf', waarbij de interne structuur zó is dat wat het eerst op die 'shelf' gezet is, er ook het eerst afgehaald wordt.

*S : Het enige verschil met *Q is dat nu het eerste wat op de 'shelf' gezet wordt, er het laatste afgehaald wordt.

*X : Nu mag niet meegegeven worden om welke elementen het gaat. De lijst ziet er dan als volgt uit: (*X naam1).
Door deze operator wordt de inhoud van de 'workspace' vervangen door de inhoud van 'shelf' naam1 en de inhoud van 'shelf' naam1 wordt vervangen door de inhoud van de 'workspace'.

Als operator kan in de 'routing section' ook voorkomen (*P x). Door deze operator wordt de inhoud van de shelves, genoemd door x, afgedrukt. Hierbij staat x voor één of meer 'shelves' namen, of voor '/'. Is x gelijk aan '/', dan wordt de inhoud van alle tot dan toe aangemaakte 'shelves' afgedrukt.

• De 'GO TO label'

We zijn nu aangekomen bij het laatste gedeelte van een METEOR-'statement'. Dit gedeelte, de 'go-to label' geeft aan hoe het programma verder verwerkt moet worden.

Als 'go-to label' kunnen optreden:

* : De volgende regel in het programma wordt uitgevoerd.

** : Er wordt één regel overgeslagen en de daarop volgende regel wordt uitgevoerd.

naam1 : Het programma springt naar de regel met als naam 'naam1'.

/ : Wordt dit teken gebruikt als 'go-to label' dan wordt dezelfde regel net zo lang uitgevoerd totdat de zoekprocedure in de 'left-half-pattern' mislukt. Het programma gaat dan automatisch naar de volgende regel.

• Print-mogelijkheden

Binnen METEOR zijn diverse mogelijkheden om de 'workspace' en eventueel andere aanduidingen te laten printen. Wil men iets laten printen, dan bestaat de desbetreffende regel uit drie gedeeltes, namelijk:

(naam1 ((*P a tekst)) naam2)

'naam1' is de naam van de regel en 'naam2' is de 'go-to label'. 'tekst' is een willekeurige tekst die al of niet samen met de 'workspace' gedrukt moet worden. Dit kan gestuurd worden met de 'a'-operator. Deze operator kan de volgende waarden aannemen:

' ' : Een spatie veroorzaakt het afdrukken van de tekst met daaronder de inhoud van de 'workspace'.

* : Nu wordt de tekst opgevat als 'comment' en wordt alleen de 'workspace' afgedrukt.

** : Het afdrukken van de 'workspace' wordt onderdrukt en alleen de tekst wordt geprint.

• Controles

Blijkt uit de executie van een programma dat het programma niet doet wat het geacht wordt te doen, dan kan men een zogenaamde 'trace' laten maken. Dit houdt in, dat van iedere regel in het programma eerst de 'workspace' geprint wordt, daarna de programmaregel en dan de veranderde 'workspace'. Om deze 'trace' te verkrijgen moet men ná de regelnaam, maar vóór de 'left-half-pattern' het element *T invoegen. Wil men echter controleren of één bepaalde regel het goed doet, dan moet niet *T maar *M ingevoegd worden. Deze 'trace' werkt alleen maar op die ene regel.

3.3 TOEPASSINGEN

De rest van dit boek zal worden besteed aan het overzicht van nieuwe toepassingen op het gebied van automatische tekstverwerking. Daarbij klimt de serie toepassingen op van letterniveau naar de hele tekst.

De meest minimale eenheid in een tekst is de letter. Bij de huidige stand van zaken in de computerlinguistiek zijn er nog maar twee probleemgebieden over die met het letterniveau te maken hebben. Voor andere letterlijke perikelen bestaat standaardprogrammatuur zoals OCP. De twee probleemgebieden zijn: automatische woordafbreking en automatische omzetting van letters in klanken. Voor deze beide probleemgebieden geldt dat de oplossingen die in het verleden werden bedacht niet in de praktijk voldoen. Voor beide problemen werden oplossingen bedacht waarvan de strategie tot onvoorspelbare resultaten leidde. Dat is natuurlijk een niet te tolereren toestand. We zullen in het kort aandacht moeten besteden aan die strategieën om te zien wat het antwoord zou kunnen zijn op de problemen. Daarna kunnen we bezien of we het instrumentarium hebben om de gewenste strategie te gaan toepassen.

3.3.1 Woordafbreken ofwel: Wat voor een teen is een baks-teen en wat voor een mokkel is een legers-mokkel

Iedere enigszins behoorlijke WORD PROCESSOR heeft een woordafbreekprogramma. Uit dit woordafbreekprogramma komen doorgaans de volgende fouten wat het Nederlands betreft:

cont-role	baks-teen
tel-exist	legers-mokkel
parap-sychologie	kwas-tje
in-itiatieven	binne-nuit
Afg-hanistan	demons-treren
ges-lepen	boods-chap
eeuw-e-noude	inkomen-sachte-ruitgang
hoge-rop	brands-tichters
niet-snut	mees-lepen
sterks-te	le-suur
ou-drussisch	bedrijf-stakken
revol-utie	rupsens-oorten
ge-int-jes	vrede-snaam
mi-slukken	

Onder programmeurs op software-houses circuleren lijsten met moeilijke gevallen. Een willekeurige greep uit zo'n moeilijke-gevallen-lijst geeft voor de letter C bijvoorbeeld:

checkout - closetafvoer - closetpapier - clubarts - clublokaal - cilindrisch - confectie - conifeer - confectiezaak - colijnsplaat - continenten - controversiële - controle - conversatie - conversatiezaal - communicatiestoornis - compromis - computer - computeronderzoek - cursusuren - cursussen.

De automatische afbreking van al deze woorden wordt door kenners voor onoplosbaar gehouden. Toch gaat het niet over zo extreem onnederlandse woorden dat men zou zeggen: "Ja natuurlijk, het is ook dwaas om op dergelijke woorden te rekenen."

Meestal beweren programmeurs van woordafbrekers dat hun programma voor 98 à 99% goede resultaten geeft. Bekijkt men het soort van onoplosbaarheden dan is duidelijk dat dit percentage van goede oplossingen niet echt met de werkelijkheid in overeenstemming kan zijn. De 98 à 99% berust op het toeval, dat van de onoplosbare gevallen er niet al te veel moesten worden afgebroken. In feite ligt het foutenpercentage natuurlijk ergens in de buurt van de 30% of meer. (Let op woorden als 'binne-nuit' of samengestelde woorden als 'clubarts'.)

Hoe zijn die fouten te verklaren? Uit het mysterieuze vermogen van de mens om uit heilige inspiratie telkens toch weer de juiste afbreking te weten? Of zou de basis van de normale woordafbreekprogramma's niet helemaal gezond zijn? Om in deze vraag enig inzicht te krijgen moeten we terug naar het lijstje met foutieve afbrekingen (die overigens allemaal werden opgetekend uit het NRC/Handelsblad). Op het eerste gezicht vertoont het foutenlijstje geen enkele regelmaat en ziet het ernaar uit dat er geen systematische fout wordt gemaakt. Zo zijn woorden als 'revol-utie' en 'binne-nuit' naast elkaar niet direct te verklaren. Het gaat daarbij immers om hetzelfde geval van woordafbreking. Een van de hoofdregels voor afbreking van Nederlandse woorden is namelijk: staat tussen twee klinkers slechts één medeklinker, dan moet worden afgebroken vóór die medeklinker. Op deze regel is echter één uitzondering, namelijk dat de medeklinker in kwestie niet bij het eerste woorddeel mag horen. In ons voorbeeld mag de 'n' van 'binnen' niet worden verschoven naar 'uit'. Wel moet de 'l' van 'revolutie' naar de 'u' van 'utie'. Een zelfde soort fout vinden we bij de afbrekingen 'vrede-soverleg', 'in-itiatieven', 'tel-exist' en 'hogerop'.

Uit deze lijst van fouten lijkt naar voren te komen dat de computer zo is geprogrammeerd dat deze regel wordt omgekeerd. Dat is echter niet een juiste verklaring, want in veel gevallen – zoals men ook in de krant van vandaag kan constateren – gaat dit soort afbreking juist wel goed. Wat kan dan wel de verklaring zijn voor deze mysterieuze foutieve afbrekingen? Om die vraag te kunnen beantwoorden moeten we aandacht besteden aan de basisstrategie die in de computerprogramma's voor woordafbreking wordt gebruikt.

• De afhap-strategie

Het belangrijkste stuk van de strategie die voor woordafbreking werd bedacht is het opstellen van lijsten waarin woorden of woorddelen zijn opgenomen. Het computerprogramma zorgt ervoor dat het af te breken woord wordt vergeleken met de woord(del)en uit die lijsten. Komt een stuk van het af te breken woord voor in een van de lijsten dan wordt het afgehapt. Moet bijvoorbeeld het woord 'narigheid' door een computerprogramma dat volgens deze strategie werkt, worden afgebroken, dan wordt eerst het stukje 'heid' afgehapt, zodat voor de 'h' van 'heid' het afbreekteken kan worden geplaatst.

Er zijn verschillende soorten lijsten. De lijst waar 'heid' in zit duidt men aan met 'de lijst van achtervoegsels'. Dit is overigens een wat ongelukkige term omdat in die lijst ook allerlei woorddelen (met de afbreking ervan) zijn opgenomen die een taalkundige nooit zou beschouwen als achtervoegsel van het soort 'heid'/'schap' enz. ... Om een voorbeeld te geven: het meest geavanceerde computerprogramma dat werkt volgens deze strategie heeft in de lijst van achtervoegsels ook woorddelen als '-avond', '-pro-gram-ma', '-in-dus-trie'. Een echt achtervoegsel is echter alleen het stukje 'trie' van industrie.

Naast de lijsten van achtervoegsels zijn er ook lijsten van 'voorvoegsels'. Deze hebben dezelfde soort kenmerken als de achtervoegsellijsten, dat wil zeggen er zitten woorddelen in die volgens taalkundige begrippen echte voorvoegsels zijn. Er zitten eveneens woorddelen in die lijsten zonder dat er een aanwijsbare taalkundige reden voor is. Het belangrijkste verschil tussen de verschillende computerprogramma's voor woordafbreking is het verschil in wat er bij de afzonderlijke programma's in de lijsten zit en de volgorde waarin de lijsten worden geraadpleegd.

Bekijken we nu het rijtje foutieve afbrekingen nog eens, dan ligt de verklaring voor een reeks van die fouten voor de hand. In de lijst van 'achtervoegsels' voor het NRC-programma zitten kennelijk ook onder andere de volgende stukjes om af te happen 'te' en 'tje'. Die veroorzaken de foutieve afbrekingen 'sterks-te', 'kwas-tje', 'belangrijks-te'. In de lijst van 'voorvoegsels' zullen wel de volgende stukjes voorkomen: 'tel', 'in', 'ge', 'vol', 'eeuw'. Daaruit kunnen dan de volgende foutieve afbrekingen worden verklaard: 'tel-exist', 'in-itiatieven', 'ge-intjes', 'eeuw-enoude', 'revol-utie'. In de lijst ontbreken kennelijk de echte voorvoegsels 'binnen' en 'hoger' waarmee dan ook de fouten 'binne-nuit' en 'hoge-rop' zijn verklaard. Van de dertig gevonden foutieve afbrekingen konden er dus tien, ofwel een derde deel, worden verklaard uit de eerste belangrijkste pijler van de strategie, namelijk de afhaptruc.

Om de andere fouten te kunnen verklaren moet de tweede belangrijke pijler in de traditionele strategie van woordafbreking per computer worden bekeken. Die pijler duiden we aan met de 'gok-strategie'.

• De gok-strategie

Het tweede basiselement van de computerprogramma's voor woordafbreking duidt men het beste aan met de term 'gokken'. Nadat de lijsten doorlopen zijn en het woord is afgehapt totdat er alleen nog maar een soort 'rompwoord' over is, gaat het programma verder zoeken naar mogelijke punten van afbreking. Een houvast daarbij is het voorkomen van klinkers. Het computerprogramma zoekt twee klinkers en bepaalt hoeveel medeklinkers daartussen voorkomen. Staat er slechts één medeklinker, dan volgt de bovengenoemde hoofdregel, dat wil zeggen de afbreekstreep komt vóór de medeklinker. Staan er twee medeklinkers tussen de twee klinkers dan wordt weer een nieuwe lijst geraadpleegd, namelijk een lijst van veel voorkomende medeklinkercombinaties. In deze lijst zitten bijvoorbeeld de combinaties 'sl', 'st', 'sn', enz. ('ou-drussisch', 'mi-slukken', 'vrede-snaam', enz.). Behoort de medeklinkercombinatie niet tot deze lijst, dan wordt tussen de twee de afbreekstreep gezet. Bij drie medeklinkers gaat het programma op dezelfde manier te werk, dat wil zeggen eerst wordt naar de combinatiemogelijkheid gekeken (denk bijvoorbeeld aan 'sch'), dan wordt de lijst weer afgelopen, enz. Behoort de combinatie niet tot de lijst van twee of de lijst van drie dan wordt gewoon iets gepakt, bijvoorbeeld 'altijd' wordt dan afgebroken na de eerste of na de tweede medeklinker. Het is duidelijk dat dit niets anders is dan in het wilde weg gokken. Deze manier van doen wordt ook verder toegepast bij vier of vijf medeklinkers na elkaar tussen twee klinkers. Dat wil zeggen: hoe meer medeklinkers in een woord na elkaar voorkomen, des te groter de kans op fouten bij de woordafbreking door het computerprogramma. Met de afhap- en gokstrategie zijn de basisideeën voor de in zwang zijnde computerprogramma's voor woordafbreking gegeven. Deze computerprogramma's zou men kunnen aanduiden met de term: programma's van de 'eerste generatie'. Het verschil tussen de programma's onderling is de inhoud van de lijsten onderling. Daarin is men tot een bepaalde graad van 'slimheid' gekomen.

• Nadelen van de afhap-gokstrategie voor woordafbreking

In het bovenstaande zijn al enige nadelen van de afhap-gok-strategie gebleken. Concreet komen ze erop neer dat de uitslag van wat het afbreekprogramma zal opleveren niet (helemaal) te voorzien is. Met andere woorden: de programma's van de eerste generatie zijn niet betrouwbaar. Vraag is dus: is de betrouwbaarheid van deze strategie op te voeren en zo ja, hoe?

Het standpunt van de programmeurs die in de afhap-gokstrategie verder werken is, dat de betrouwbaarheid wel degelijk op te voeren is. Het middel daarbij is: een andere en 'slimmere' vulling van de lijsten. Ofschoon al zo'n jaar of 15 aan en met deze strategie wordt gewerkt, moet worden vastgesteld dat het niet lukt het basisgebrek,

namelijk de in principe onvoorspelbaarheid der resultaten op te lossen. Het blijkt namelijk steeds meer dat een verbetering op de ene plaats fouten op een andere plaats veroorzaakt. Dit ligt ook geheel in de lijn der verwachtingen. Een van de fundamenten van de strategie is namelijk gokken.

Aan het eind van de pogingen om het programma te verbeteren door de lijsten te veranderen/uit te breiden staat bovendien dat men een compleet woordenboek met correct afgebroken woorden in de computer inbrengt en het programma alleen nog maar snel laat zoeken en vergelijken. Ook deze oplossing echter loopt voorspelbaar vast, zeker voor een taal als Nederlands, omdat er steeds weer nieuwe woorden bijkomen (juist in de krant) en dus toch weer moet worden teruggegrepen op de oude onbetrouwbare afhap-gokstrategie. De belangrijkste nadelen van de eerste generatie afbreekprogramma's zijn dus samen te vatten met de begrippen: onbetrouwbaarheid en onverbeterbaarheid.

• Tweede generatie afbreekprogramma's

Het belang van de eerste generatie programma's is dat men er grondig van kan leren hoe men dit soort problemen niet kan oplossen. Er is echter meer, want deze ontdekking dwingt tot een bezinning op de basisfilosofie die achter dit soort programmatuur zit en die welke er achter zou moeten zitten. Als zodanig kunnen de programma's die we nu hebben tot een echte doorbraak leiden, wat ze dan ook gedaan hebben in de tweede generatie.

De verklaring waarom de afhap-gokstrategie gekozen werd is daarbij van groot belang. Deze strategie is zeker niet gekozen op grond van kennis van de (Nederlandse) taal alleen. Evenmin berust de strategie op taalkundig inzicht alleen. Integendeel: in de lijsten bijvoorbeeld zitten allerlei woorddelen die in de taalkunde als zodanig niet zouden worden erkend. Vanuit de taalkunde gezien zitter er allerlei willekeurige elementen in die lijsten. Net zo min kan het de bedoeling van de programmamakers zijn geweest de woordafbreekstrategie van de gemiddelde Nederlander na te bootsen. Het lijkt namelijk niet erg waarschijnlijk dat een Nederlander voor woordafbreking een serie lijsten in welke volgorde dan ook zou moeten afwerken voor hij tot een beslissing komt. Integendeel, de lijstenstrategie wordt volledig veroorzaakt door het instrument dat men gebruikt.

Is nu dan het verlossende woord gevonden en het bewijs dat de computer zoveel van de mens verschilt dat het onmogelijk is, menselijke taken te programmeren? Het antwoord is gladweg: nee. Men is namelijk helemaal niet verplicht in lijststructuren te denken wanneer men computers gebruikt en programmeert. Alleen wanneer men een bepaald type programmeertaal gebruikt moet men in 'lijsten' of liever in 'reeksen, rijen' denken. Nu is de meerderheid van de in Nederland via de wiskunde opgeleide programmeurs helemaal ingebed in het 'lijstdenken'.

Daar ligt dan ook de verklaring dat de informaticus met deze achtergrond gemakkelijk naar de verkeerde middelen grijpt. Rijen zijn heel handig voor allerlei problemen die met getallen te maken hebben. Voor computerbewerkingen van taal zijn het ondingen, zoals zonneklaar blijkt uit de eerste generatie computerprogramma's voor woordafbreking. Eenvoudig gezegd: wanneer iets geen 'lijstenprobleem' is, is het onverstandig het met gebruikmaking van rijen op te lossen. Ofwel: problemen van taalkundige aard vereisen een ander soort informatica dan problemen van rekenkundige aard. Wat moet men dan wel gebruiken om het probleem van de woordafbreking op te lossen? Die vraag kan men alleen beantwoorden wanneer men een idee heeft hoe de mens in principe dit soort taken verricht. Men moet dus aansluiting zoeken bij de psychologie, in dit geval bij het onderdeel 'kunstmatige intelligentie'. Daarachter echter staat voor het probleem van woordafbreking een onderdeel uit de taalpsychologie, namelijk taalperceptie of taalwaarneming. Om een voorbeeld te nemen: stel, een Nederlander moet uitmaken waar hij de volgende serie woorden moet afbreken aan het begin:

-onteren	-onthouden
-ontegenzeggelijk	-ontheologisch

De eerste hypothese is: je kent de woorden 'eren', 'tegen', 'houden' en 'theologisch', dus afbreken vóór het begin van die woorden. Aan deze hypothese ligt het idee ten grondslag dat een taalgebruiker zijn 'innerlijk' woordenboek raadpleegt bij woordafbreking. Met die hypothese zijn we aardig dicht in de buurt van een 'lijstenstrategie'.

Is het echter noodzakelijk voor een kenner van het Nederlands om altijd in zijn 'innerlijk' woordenboek na te gaan welke woorden hij kent? Of zou hij ook door zijn kennis en dagelijks gebruik van het Nederlands zo vertrouwd zijn met de taal dat hij een verwachtingspatroon heeft opgebouwd op grond waarvan hij letter- en klankpatronen herkent? Inderdaad is er enige wetenschappelijke evidentie dat de taalgebruiker met verwachtingspatronen werkt bij de waarneming van taal. Verwachtingspatronen nu kunnen worden 'geleerd' aan een computer. Men bevindt zich dan op het gebied van de zogenaamde 'patroonherkenning'. Uit het lijstje voorbeelden springt bijvoorbeeld direct een 'onmogelijk' patroon in het oog, namelijk 'heol'. Beschikt het computerprogramma over de kennis dat geen woord in het Nederlands met 'heol' kan beginnen (er is er wel een dat begint met 'heor', namelijk 'heortologie', de leer van de kerkelijke feesten), dan is de afbreking 'on-theologisch' geregeld. Men hoeft daarbij niets te gokken, evenmin ontstaat het probleem dat men niet meer weet welke lijst op welk punt wat heeft veroorzaakt.

EEN PATROONHERKENNEND PROGRAMMA VOOR WOORDAFBREKING

Als voorbeeld hoe een patroonherkennend programma voor woordafbreking gemaakt kan worden volgt hier een programma in METEOR dat de problemen oplost van het rijtje met in het NRC gevonden fouten.

- Taalkundige regels die nodig zijn

De belangrijkste regels voor woordafbreking in het Nederlands zijn de volgende drie:

1. Staat tussen twee klinkers één enkele medeklinker dan moet worden afgebroken voor de medeklinker.
   Voorbeeld uit het rijtje: re-vo-lu-tie
2. Staan tussen twee klinkers twee medeklinkers dan moet tussen de twee medeklinkers worden afgebroken.
   Voorbeeld uit het rijtje: vredes-naam
3. Staan twee klinkers naast elkaar dan moet tussen die twee klinkers worden afgebroken.
   Voorbeeld uit het rijtje: initi-atieven

Om deze hoofdregels te programmeren zou het handig zijn klinkers en medeklinkers in twee klassen te verdelen en wel zo dat direct herkenbaar is of medeklinkers worden omgeven door klinkers. Daarbij willen we ook de individuele waarde van de klinkers zelf niet vergeten, want die hebben we later weer nodig bij de bepaling van uitzonderingspatronen. Zo moeten we weten dat de combinatie: -nenuit moet worden afgebroken na de tweede n.

Om dit te kunnen bereiken geven we de klinkers een cijferwaarde. Dan kunnen we de patronen controleren met de match-opdracht

($. NUMBER) in de linkerhelft van de regel.

De volgende klinkers komen in de voorbeelden voor:

o e i a ie ij eeu ou u ui oo aa

Natuurlijk moeten eerst de langste klinkers gecodeerd worden anders zou namelijk de klinker 'aa' bijvoorbeeld worden gelezen als tweemaal een losse 'a', waarop hij dan later zou worden afgebroken.

De cijferwaarden voor de verschillende klinkers zijn vanzelfsprekend willekeurig. De METEOR regels kunnen er met deze specificaties nu als volgt uitzien:

```
(*    (AA)    (1)    /)
(*    (EEU)   (2)    /)
(*    (OU)    (3)    /)
(*    (IE)    (4)    /)
(*    (IJ)    (5)    /)
```

```
(*    (UI)     (6)    /)
(*    (OO)     (7)    /)
(*    (A)      (8)    /)
(*    (E)      (9)    /)
(*    (I)      (10)   /)
(*    (O)      (11)   /)
(*    (U)      (12)   /)
```

De hoofdregels kunnen nu als volgt geprogrammeerd worden:

```
(*   ( ($.NUMBER)    ($.NUMBER) )    (1 (*DASH) 2 )     /)
(*   ( ($.NUMBER)  ($.1)  ($.NUMBER)  )  (1 (*DASH)2 3 )   /)
(*   ( ($.NUMBER)  ($.1)  ($.1)  ($.NUMBER) )  (1 2 (*DASH) 3 4 )
```

VRAAG. Zou men in de laatste regels ook hebben kunnen schrijven in plaats van het patroon ($.1) ($.1)
het patroon ($.2)?

TENZIJ

- Uitzonderingen op de hoofdregels

1. Een medeklinker

Er is een serie patronen die niet volgens de hoofdregel gaat. Tegen de hoofdregel van een medeklinker tussen twee klinkers in gaan gedeeltes van bijvoorbeeld het woord eeuwenoude of hogerop.

Het patroon 'enoude' of het patroon 'erop' bevat een afwijkende afbreekverplichting. Het is geen kunst deze twee patronen op te geven vóór de hoofdregel, als volgt:

```
(*    ( 9 N 3 )    ( 1 2 (* DASH )    3)     *)
(*    ( 9 R 11 P)  ( 1 2 (* DASH )  3 4)     *)
```

OPGAVE. Schrijf de METEOR regels voor de andere uitzonderingen op de hoofdregel van een medeklinker tussen twee klinkers.

2. Twee medeklinkers

In het lijstje foute afbrekingen komen ook fouten voor die worden veroorzaakt door de regel aangaande twee medeklinkers tussen twee klinkers. Bijvoorbeeld het woord 'parapsychologie' en het woord 'gesteven'.

Om deze twee woorden correct afgebroken te krijgen kan men verschillende dingen doen. Men zou kunnen denken in voorvoegsels als 'para' en 'ge'. Dan echter moet men wel de uitzonderingen op die regels eveneens voorzien. In dit geval ligt dat verschillend bij de twee woorden. Para als voorvoegsel is zeer constant. Alleen parallel is

een uitzondering. Reden waarom 'para' best als patroon kan worden opgegeven, voorafgegaan door parall. Aldus:

```
(*    ( P A R A L L )        VR)
(*    ( P A R A ($ 1) )    ( 1 2 3 4 (* DASH ) 5 )    *)
(VR     (     )                (     )                *)
```

(Denk echter aan 'paraat'!)

Het patroon 'ge' is aanzienlijk moeilijker. Een oppervlakkige blik in een woordenboek leert al dat er zulke woorden zijn als 'gemzeleer', 'gerst', 'geen', 'gentiaan', 'genre', 'gent'. Zo te zien verdient het niet sterke aanbeveling om 'ge' op te geven als patroon. In ieder geval moet er een uitgebreide detailstudie naar gedaan worden. Anders lopen we zeker de kans allerlei avontuurlijke afbrekingen te krijgen. Duidelijk is dat dit probleem alleen kan worden opgelost door taalkundige studie. Het kan niet zomaar uit het losse handje worden geprogrammeerd. De veiligste oplossing voor ons geval is om dan maar dit concrete patroon 'gest' op te geven.

```
(*   (G 9 S T )     ( 1 2 (* DASH ) 3 4 )     *)
```

Van de woorden die beginnen met 'gest' wordt nu dus altijd de 'ge' afgenomen. Ook woorden als 'gestreken' zijn correct opgelost. Niet opgelost is het woordpaar: gestel/Gestel (in St. Michielsgestel). Dit kan ook niet worden opgelost met patrooninformatie alleen. Men moet namelijk weten dat Gestel een eigennaam is, dan is de oplossing gegeven. Hoe eigennamen worden ontdekt wordt in een andere paragraaf behandeld.

OPGAVE. Los ook de andere uitzonderingsgevallen op de twee-medeklinker-regel op.

3. Drie medeklinkers

Niet in de hoofdregels vallen de woorden waarin drie (of meer) medeklinkers naast elkaar tussen twee klinkers voorkomen. Ook in deze groep komen in de normale woordafbreekprogramma's veel fouten voor, omdat hier het gok-gedeelte van de strategie wordt toegepast. Men kan het probleem van het woord met meer dan twee medeklinkers naast elkaar vanuit twee gezichtshoeken oplossen. Ten eerste kan men zoeken naar medeklinkercombinaties die in het Nederlands niet binnen een woord kunnen voorkomen. Verder kan men speciale regels opgeven voor combinaties van medeklinkers die in iedere samenstelling kunnen voorkomen. Een voorbeeld van niet-combineerbare medeklinkers in een woord is de combinatie 'fm' in het woord zelfmedelijden. De METEOR regel voor deze afbreekregel is:

(* (F M) (1 (* DASH) 2) *)

OPGAVE. Zoek de andere niet-combineerbare patronen op en schrijf de METEOR regels ervoor.

De tweede optiek wordt toegepast op de overblijvende gevallen. Deze optiek hield in dat iedere combinatie van medeklinkers mogelijk was. Een voorbeeld daarvan is de combinatie T S N in nietsnut: zowel TS als SN is een combinatie die is toegestaan in Nederlandse woorden, de een aan het eind, de ander aan het begin van een woord.

Het woord nietsnut kan men alleen goed afgebroken krijgen wanneer bekend is dat voor de TS de klinker 'IE' staat. De regel in METEOR ziet er dus als volgt uit:

(* (4 T S N) (1 2 3 (* DASH) 4) *)

OPGAVE. Schrijf voor de overblijvende gevallen de bijbehorende METEOR regels.

- Welke woorden kan een patroonbreker niet aan?

In principe is het vrij eenvoudig gebleken met een tweede generatie computerprogramma's de 'onverklaarbare' fouten uit de eerste generatie (denk aan: revol-utie naast binne-nuit) op te lossen. De tweede generatie werkt zonder woordenboek, dus ook zonder woordbetekenis. Dat wil zeggen: zodra alleen op grond van woordbetekenis kan worden afgebroken, zodra dus ook aannemelijk is dat de mens zelf zijn 'innerlijk woordenboek' echt moet gebruiken, schieten de patroonbrekers tekort. Voorbeelden hiervan zijn: balletje (een kleine bal of een klein ballet); palletje (van pal of pallet); uitje (van ui of uit); pootje (van po of van poot); enz. In dit geval kiest een patroonbreker een van de twee en dus mogelijk het verkeerde woord. De woordafbreking zelf echter is correct en niet bizar zoals zo dikwijls bij de afhap-gok-strategie het geval was.

Wil men ook deze woorden correct oplossen, dan zal men de computer meer moeten leren van het Nederlands, namelijk de betekenis van woorden. Hiermee zou de computer in staat gesteld zijn weer een stukje meer van 'intelligent menselijk gedrag' na te bootsen. Dat is namelijk wat men doet wanneer men computers taken laat verrichten die met taal te maken hebben. Of anders gezegd: het samengaan van taal(kunde) en informatica leidt noodzakelijkerwijs naar de psychologie, namelijk naar de 'kunstmatige intelligentie', de wetenschap van nabootsing met computers van 'intelligent' gedrag. Neemt men een andere strategie, bijvoorbeeld een puur op computerkunde gebaseerde strategie – dat leert bijvoorbeeld de eerste generatie woordafbrekers – dan komt men voor onoplosbare problemen te staan.

In een latere paragraaf zal blijken dat hetzelfde geldt voor pro-

gramma's die zich puur naar de 'taalkunde' richten. De conclusie die daar dikwijls uit getrokken wordt is, dat 'taal' en 'computer' niet samen gaan en dat de computer dus ongeschikt is voor taal; een conclusie die te haastig werd getrokken zoals de tweede generatie woordafbrekers bijvoorbeeld (maar ook andere toepassingen) duidelijk maakt.

3.3.1.1 Als letters klinken moeten

- De droom van een spraakmachine

Omstreeks 1788 bouwde Wolfgang von Kempelen (geb. 23-1-1734) een machine die de stem van een drie- tot vierjarig kind kon nabootsen. In 1821 werd deze machine die Von Kempelen beschreef in "Mechanismus der menslichen Sprache" door een zekere Posch in Berlijn verbeterd en nagebouwd. Ook op het Instituut voor Fonetiek van de Universiteit van Amsterdam bestaat tegenwoordig een nabootsing van de spraakmachine van Von Kempelen. Von Kempelen gaat echter door voor een bedrieger en een fantast, ook al overleed hij als ambtenaar aan de Weense Kanselarij. Deze slechte naam heeft hij ook op het gebied van de spraakmachine.

De Amsterdamse nabootsing schijnt volgens de bouwers ook niet veel meer te kunnen dan een serie medeklinkers uitspreken. Een late nagalm van de mechanische spraakmachine werd gedemonstreerd op de eerste internationale workshop voor natuurlijke communicatie in Warschau (september 1980). Op het Instituut voor Cybernetica van de Universiteit van Tiflis werd een machine gemaakt die wel in staat blijkt luid en duidelijk zinnen uit te spreken.

De machine van Von Kempelen was een mechanische machine, niet een machine die zelf iets kon samenstellen uit gegevens die op een of andere wijze worden ingevoerd zoals dat gaat bij de moderne computer. De moderne techniek die van de computer gebruik maakt bij het vormen van spraakklanken noemt men 'spraaksynthese'. In Amerika kan men voor nog geen $ 400 een spraaksynthese-computer kopen. Men zou enigszins optimistisch kunnen stellen dat het probleem van de spraaksynthese in vergaande staat van oplossing verkeert.

Een andere machine, en hier ziet men plots het maatschappelijk belang van een spraakmachine, werd onlangs aangekondigd in Amerika, een machine die gedrukte tekst kan lezen en omzetten in gesproken tekst. Deze machine maakt een elektronische foto van de gelezen letters, voegt die samen tot woorden, zet ze om in een code voor de uitspraak en geeft dan de spraaksynthese voor de gevonden woorden. Volgens het prospectus gebruikt de minicomputer voor het omzetten van het gelezen woord naar de erbij behorende uitspraakcode voor het Engels meer dan 1.000 taalkundige regels. De blinden die de machine gebruiken schijnen er heel content mee te zijn.

Op zich is het dus mogelijk een machine te maken die kan voorlezen. Het grootste probleem daarbij is niet meer de spraaksynthese, maar de omzetting van het schriftbeeld in het klankbeeld. Dit is uiteraard per taal verschillend. Zo ligt bijvoorbeeld voor het Nederlands de uitspraak van 'vlucht' vast. Het maakt niet uit of er staat 'hij vlucht' of 'een vlucht duiven'. In het Duits echter bestaan er twee woorden die gespeld worden als 'Flucht'. Het ene is van het werkwoord fluchen (vloeken) (uitgesproken met een lange oe), het andere betekent vlucht (van de op vlucht slaan) (uitspraak korte oe).

- Een voorleesmachine voor het Nederlands

Om een goede voorleesmachine te maken moet men een computerprogramma ontwerpen dat geschreven woorden zo codeert dat die code kan worden gebruikt door een spraaksynthese-programma. Verder moet er een computerprogramma geschreven worden dat de klemtoon van de woorden en de zinsmelodie alsmede de aanpassingen in klank van verschillende woorden aan elkaar binnen een zin vindt. Omdat de omzetting schrift/klank van het woord het moeilijkste gedeelte is, behandelen we verder enige problemen die hier opdoemen.

- Letters zijn niet eenduidig

Voor een machine, net als voor een kind dat pas begint te lezen, is het niet duidelijk dat dezelfde letters soms toch anders worden uitgesproken. Voorbeelden daarvan zijn: de letter 'g' in 'dag-dagen', of de letter 'd' in 'bed-bedden'. De 'p' in 'pet' wordt anders uitgesproken dan de 'p' in 'opdoen', waar men in feite een 'b' hoort, maar de 'b' in 'clubkas' wordt als 'p' uitgesproken en de 'd' in 'badgast' als een 't'. Ook komt het regelmatig voor dat er wel een letter wordt geschreven maar dat daarmee geen klank correspondeert. Bijvoorbeeld de 'd' in 'kinds', de 't' in 'onverwachts'. Een extra mooi geval is het woord 'herfstdag', waar eerst de 't' verdwijnt en dan de 's' wordt uitgesproken als 'z' onder invloed van de op de 's' volgende 'd'.

Bovengenoemde gevallen van klank-/schriftmutaties zijn echter aan regels onderhevig zodat er een computerprogramma voor kan worden geschreven waarin die regels zijn vervat. Zo is bijvoorbeeld een zogenaamde stemhebbende medeklinker aan het eind van een woord altijd stemloos ('b' wordt 'p' en 'd' wordt 't'). Twee gelijke medeklinkers worden in het Nederlands maar één keer uitgesproken. De computer kan een dubbele medeklinker dus gewoon enkel maken. Wordt aan het eind van een woord of woorddeel 'ng' geschreven, dan is dat één klank (bang). De linker medeklinker bij een opeenvolging van twee krijgt het karakter van de rechter medeklinker (zie 'herfstdag' en 'zakdoek').

Ingewikkelder problemen ontstaan bij de letter 'e'. Naast elkaar bestaan de woorden tafel, tabel, kabel en libel. Het woord 'evenredig' bevat drie 'e's, waarvan de tweede wordt uitgesproken als een 'u'

(van 'put'). Een woord als 'beregoed' wordt, wat de eerste twee 'e'-klanken betreft, anders uitgesproken dan het woord 'beregelen'. Het woord 'beregelen' is overigens ook aardig in vergelijking met 'even-redig'. Het kost geen moeite tientallen voorbeelden te vinden van moeilijkheden met het beregelen van de klank die moet worden verbonden aan de geschreven 'e'.

De '-ig' is ook aardig als illustratie van de moeilijkheden die een programmeur ondervindt wanneer hij een voorleesmachine wil maken. Men vergelijke: vredig met liggen en spuigaten. Dat laatste lijkt voor de normale lezer een grapje, maar voor de computer is het een serieus probleem. In het programma moet een regel worden opgenomen zodat de computer weet dat eerst moet worden bekenen of de 'i' niet bij de tweeklank 'ui' hoort.

Met dit laatste probleem is weer een nieuwe categorie van moeilijkheden in het vizier gekomen, namelijk het probleem dat vooral speelt bij samengestelde woorden. Voorbeelden zijn (geredeneerd vanuit de computer): gasexplosie en lesuur. Pas wanneer de computer geleerd heeft dat het niet gaat om ga-sex-plosie, maar om gas-ex-plosie, kan hij beslissen dat de 'a' in het eerste deel kort is. Hetzelfde geldt voor de 'e' in lesuur, dat niet komt van le-suur, maar van les-uur. Wanneer men zich herinnert hoeveel moeite het heeft gekost om voor woordafbreking een strategie te vinden die juist dit soort problemen oplost, is in één klap duidelijk hoe ingewikkeld het moet zijn een leesmachine voor blinden te maken.

Er zijn echter nog meer problemen. Ze worden duidelijk in de volgende woorden: 'vrouwelijk', 'bommelding' en 'gelig'. De meeste lezers van Nederlands zullen direct één mogelijkheid kiezen bij 'vrouwelijk' en 'gelig'. Velen zullen zich eerst afvragen wat voor een ding een 'bommelding' is, tot ze tot de conclusie komen dat het niet gaat om een ding, maar om een melding. 'Bommelding' ligt daarmee op de lijn van 'spuigaten' en 'gasexplosie', zij het dat het dit keer om een probleem gaat dat ook door een normale taalgebruiker wordt herkend en niet alleen wordt veroorzaakt door de onnozelheid van de computer. De twee andere woorden vertonen een echte meerduidigheid die men alleen kan oplossen wanneer men de uitspraak weet:

- Het vrouwelijk deel van de schepping begint zich in sommige culturen bewust te worden van haar 'onderdrukte' positie.
- In de sloot lag een onherkenbaar vrouwelijk.
- Het voorwerp was gelig van kleur.
- Al dat gelig ben ik nu zat.

Tot slot heeft men nog de uitgebreide categorie van moeilijkheden in woorden die uit vreemde talen komen. De Nederlander houdt bij voorkeur de oorspronkelijke klank aan en moet leren dat 'journaal' anders wordt uitgesproken dan 'Joure' en dat 'vlaai' anders klinkt dan 'play'.

Hoe kan men de computer nu deze moeilijkheden laten oplossen? Of moet men hem laten signaleren dat hij extra informatie van de

programmeur wil hebben? Misschien een lijst van uitzonderingen opstellen die later met de hand moeten worden gedaan?

Kiest men voor handwerk, dan ontstaat de moeilijkheid dat men op den duur niet meer weet wat het computerprogramma nu wel en wat het niet kan. Men construeert het programma dan namelijk al gauw zo dat in feite alle output met de hand moet worden gecontroleerd (denk aan 'tafel' en 'tabel'). Kiest men voor een volledig automatische oplossing dan lijken woorden als 'vrouwelijk' en 'gelig' onoplosbaar. Immers, zo is dikwijls de redenering, om dit soort woorden te herkennen moet de computer alles van de hele wereld weten. Toch ligt de oplossing van het vrouwelijk mysterie veel dichter bij. Als 'lijk' is 'vrouwelijk' een zelfstandig naamwoord, als deel der schepping is 'vrouwelijk' een bijvoeglijk naamwoord. De oplossing van het probleem is dus: schrijf een computerprogramma dat woordsoorten onderkent. Een dergelijk programma wordt in een latere paragraaf aan de orde gesteld.

Een kleine restcategorie van moeilijkheden blijft nu nog over. Ze worden duidelijk in het woord 'balletje' dat zowel een kleine bal als een klein ballet kan aanduiden. Bij dit soort woorden gaat de truc van de woordklassen niet op, omdat in beide betekenissen een zelfstandig naamwoord wordt benoemd. De computer moet nu inderdaad kennis verzamelen over 'bal' en 'ballet'. Het gaat hier om een puur inhoudelijk (semantisch) probleem waarbij alleen (ook voor de normale lezer) de rest van de tekst waarin het woord voorkomt de oplossing kan brengen. Ook dit kan in principe worden opgelost: er zijn computerprogramma's geschreven die de inhoud van een verhaaltje kunnen vinden (zie verderop). Voorleesmachines zoals ze tot nu toe in ontwikkeling zijn beperken zich echter tot de gevallen waarin inhoudelijke informatie niet nodig is. Dit omdat het vinden van inhouden een totaal ander soort computerprogramma vereist en de meeste moeilijkheden toch met eenvoudiger middelen kunnen worden opgelost.

- Programma's

De oudste programma's voor het omzetten van letters in klanken zijn gebaseerd op de eerste generatie computerprogramma's voor woordafbreking. Deze programma's zijn niet erg betrouwbaar, zoals werd uiteengezet in de vorige paragraaf. Verder worden de problemen die samenhangen met de 'e' (van het soort 'tafel'/'tabel') niet automatisch opgelost.

Ook hier biedt weer een patroonherkennend programma de oplossing. Voor het Nederlands werd er zelfs een ontwikkeld. Daarbij bleken problemen als 'tafel'/'tabel' en 'beregelen'/'beregoed' wel degelijk oplosbaar zonder handwerk. Achter de strategie van patroonherkenning steekt de gedachte dat men daarmee beter de manier van de mens benadert. Dit in tegenstelling tot de eerste generatie programmatuur die meer vanuit de computer is gedacht.

Mocht het lukken een goed programma te krijgen dat geschreven tekst in gesproken tekst omzet, dan zijn er nog verdere toepassingen dan de voorleesmachine alleen. Het programma zou kunnen worden gebruikt in het lees- en spelonderwijs; het zou kunnen worden ingezet bij kinderen met woordblindheid. Verder zou het programma ook op puur wetenschappelijk gebied zijn diensten bewijzen. Er zijn namelijk nog steeds geen exacte gegevens bekend over hoe dikwijls een bepaalde klank of klankcombinatie in het Nederlands voorkomt. Toch zou het zeer wenselijk zijn over die gegevens te beschikken, al was het alleen maar ten behoeve van het onderwijs aan buitenlanders.

• Letters en klanken: het programma FONGRAF

In het Nederlands bestaan de klanken die in onderstaande tabel zijn opgenomen. Om welke klanken het gaat kan men afleiden uit de voorbeeldwoorden. De klanken werden weergegeven volgens het fonetische alfabet (kolom 1) en volgens de notatie die het computerprogramma FONGRAF oplevert. Uit de tabel kan men lezen dat de verhouding tussen letter en klank niet één op één is, wat ook al duidelijk was uit de inleidende tekst van deze paragraaf. Het programma FONGRAF werd geschreven in PL/I. Het bevat momenteel ruim 5.000 opdrachten en het werkt correct op de 1.000 meest frequente woorden van het Nederlands. Van de rest wordt meer dan 90% juist getranscribeerd. Een test op een willekeurige krantetekst leverde een foutenpercentage van slechts $1\frac{1}{2}$% op. Het is niet doenlijk het hele programma hier te herhalen. Trouwens, het programma werd uitgebreid beschreven en ook de programmatekst zelf werd gepubliceerd. Wel is het doenlijk in het kort de strategie van het programma te schetsen. Deze is redelijk goed afleidbaar uit de hoofdroutine die alle andere onderprogramma's aanroept.

PROGRAMMA FONGRAF

```
FONGRAF:
STMT  BLK    DO     SOURCE

  1     0             FONGRAF:
                       PROC OPTIONS(MAIN);
  2     1              DCL
                           WRD CHAR(25),
                           WRDA(25) CHAR(1) DEE WRD POS(1),
                           WRDB(25) CHAR(1),
                           TWRD CHAR(25) DEE WRDB,
                           VOC(6) CHAR(1) INIT('A','E','I','O','U','Y'),
                           VOB(3) CHAR(1) INIT('A','O','U'),
                           CONS(4) CHAR(1) INIT('L','M','N','R'),
                           CONSA(6) CHAR(1) INIT('F','S','L','M','N','R'),
                           CONSB(8) CHAR(1) INIT('B','D','G','J','R','T','V','W'),
                           DONS(3) CHAR(1) INIT(' ','D','S'),
                           BONS(5) CHAR(1) INIT('B','F','G','R','V'),
                           LONS(4) CHAR(1) INIT('L','N','R','W'),
                           FVW(3) CHAR(1) INIT('F','V','W'),
                           PTK(3) CHAR(1) INIT('P','T','K'),
                               (I,K,KLIJK,M) FIXED BIN STATIC EXT,
                           LVOK FIXED BIN,
                           LET CHAR(1);
  3     1                  ON ENDFILE(PAK) GOTO UIT;
                          /* INPUT KAN WORDEN GELEZEN VAN "CELAN" UIT BOOTEXT  */
  5     1                  OPEN FILE(PAK) STREAM INPUT LINESIZE(50),
                               FILE(B) STREAM OUTPUT LINESIZE (50);
  6     1                  L=0;
  7     1                  LA=0;
  8     1                IN:
  9     1                  DO M=1 TO 25;   WRDB(M)=' ';   END;
 11     1                  GET FILE(PAK) EDIT (WRD) (A(25));   GET SKIP FILE(PAK);
 13     1                  IP=INDEX(WRD,' ');
 14     1                  IF IP=1 THEN GOTO IN;
 16     1                  K,KLIJK,M=0;
 17     1                  DO I=1 TO IP;
 18     1     1            M=M+1;
 19     1     1                 LET=WRDA(I);
 20     1     1                 IF LET='D' THEN CALL BED;
 22     1     1                 IF LET='T' THEN CALL TIE;
 24     1     1                 IF LET='N' THEN CALL ENG;
 26     1     1                 IF LET='G' THEN CALL OPGANG;
 28     1     1                 IF LET='B' THEN CALL OPBOD;
 30     1     1                 IF LET='T' THEN CALL WITBOEK;
 32     1     1                 IF LET='F' THEN CALL AFDOEN;
 34     1     1                 IF LET='K' THEN CALL ZAKDOEK;
 36     1     1                 IF LET='S' THEN CALL HUISDEUR;
 38     1     1                 IF LET='V' THEN CALL OPVANG;
 40     1     1                 IF LET='Z' THEN CALL OPZIJ;
 42     1     1                 IF LET='J' THEN CALL ZJ;
 44     1     1                 IF LET='B' THEN CALL WEB;
 46     1     1                 IF LET='F' THEN CALL CODE;
 48     1     1                 IF LET='Y' THEN LET='I';
 50     1     1                 IF LET='I' THEN CALL EIER;
 52     1     1                 IF LET='O' THEN CALL CIRCA;
 54     1     1                 IF LET='W' THEN CALL ERWT;
 56     1     1                 IF LET='5' THEN CALL AAI;
 58     1     1                 IF LET='U' THEN CALL OUD;
 60     1     1                 IF LET='O' THEN CALL OIR;
 62     1     1                 IF LET='A' THEN CALL AIR;
 64     1     1                 IF LET='Q' THEN CALL AQUA;
 66     1     1                 IF LET='X' THEN CALL XENDN;
 68     1     1                 CALL VOK;
 69     1     1                 IF LVOK=0 THEN CALL ENKLI;
 71     1     1                 IF LVOK=1 THEN DO;
 73     1     2                      LET=WRDA(I+1);
 74     1     2                      IF LET=' ' THEN CALL BLANC;
 76     1     2                      IF LET='X' THEN CALL ALEX;
 78     1     2                      CALL VOK;
 79     1     2                           IF LVOK=0 THEN DO;
 81     1     3                           LET=WRDA(I+2);
 82     1     3                           IF LET='R' THEN CALL ABRI;
 84     1     3                           CALL VOK;
 85     1     3                                IF LVOK=1 THEN CALL LVOC;
 87     1     3                                          END;
 88     1     2                           END;
 89     1     1                 WRDB(M)=WRDA(I);
 90     1     1               FLAP:
                                END;
 91     1                      IF K=0 THEN CALL ALFA;
 93     1                      PUT FILE(B) SKIP EDIT(WRD,TWRD)(2(A));
 94     1                      L=L+1;
 95     1                      GOTO IN;
 96     1                    END FONGRAF;
```

TABEL VAN KLANKEN EN LETTERS IN HET NEDERLANDS

STANDAARD FONETISCHE NOTATIE	ONZE NOTATIE	VOORBEELDEN
/h/	/h/	hoed
/p/	/p/	paal
/b/	/b/	boek
/m/	/m/	maan
/w/	/w/	waar
/f/	/f/	fit
/v/	/v/	vier
/t/	/t/	teen
/d/	/d/	duim
/n/	/n/	noot
/l/	/l/	lot
/r/	/r/	rat
/s/	/s/	soep
/z/	/z/	zeep
/j/	/j/	jas
/k/	/k/	kus
/ɣ/	/X/	bagger
/X/	/x/	lachen
/n/	/ml/	aanvang
/ŋ/	/nl/	behang
/g/	/q/	bakboord
/i/	/ie/	ziel
/u/	/uu/	utrecht
/u/	/oe/	koe
/ɪ/	/i/	mis
/e/	/ee/	geel
/ʌ/	/u/	nul
/ö/	/eu/	neus
/o/	/oo/	boot
/ə/	/ə/	we
/ɛ/	/e/	vet
/ɛ:/	/e:/	air
/ɔ/	/o/	zot
/ɑ/	/a/	zat
/a/	/aa/	maat

Uit: Boot/Maat/Renkers, 1980.

Vanaf opdracht 19 tot 90 ziet men wat er gebeurt: de machine zoekt uit om welke letter het gaat en de letter zelf bepaalt welke subroutine moet worden aangeroepen. De subroutines zelf bestaan uit patroonherkenners. Dit is het grote verschil met de andere programma's voor verklanking van letters. De patroonherkenner maakt het inschakelen van woordafbreekprogramma's overbodig. Daardoor ook zijn bij FONGRAF de fouten wel begrijpelijk en ook herstelbaar. Daardoor ook is het programma zoveel succesvoller gebleken.

- Patroonherkennende regels in METEOR voor fonematisering

Patroonherkennende programma's zijn veel gemakkelijker te schrijven in METEOR dan in PL/1. We zullen dus een indruk geven van het soort regels dat men kan gebruiken om een betrouwbaar omzettingsprogramma van letter naar klank te krijgen in METEOR regels.

'eren'
Veel woorden in het Nederlands gaan uit op 'eren'. Woorden zoals 'beren', 'leren', 'speren', 'kleren', enz. Deze woorden worden uitgesproken als 'eerɣ'. Daarnaast bestaan echter woorden die ook op 'eren' uitgaan maar worden uitgesproken als ɣrɣ, bijvoorbeeld 'kinderen hinderen'. De regels hiervoor zouden in METEOR zijn:

```
(* ( N D E R E N )    ( 1 2 ɣ r ɣ )     *)
(* ( E R E N ($.0) )  ( 1 1 2 ɣ   )     *)
```

'ge'
Aan het begin van een woord wordt de combinatie 'ge' meestal uitgesproken als 'gɣ'.

In de paragraaf over woordafbreken zijn we enkele voorbeelden tegengekomen waaruit bleek hoe ingewikkeld het probleem van de uitspraak van 'ge' is. Wanneer we afzien van problemen als 'geen' waarin zich ook het patroontje 'ge' bevindt, blijven over de woorden: gentiaan, gemzeleer, gent en genre.

VRAAG. Hoe is het te beregelen in METEOR dat patronen als 'geen' geen probleem opleveren? Schrijf de METEOR regel die dit probleem oplost.

- Een gentiaan van gemzeleer: programmering in METEOR

Wanneer we de patronen die de uitzondering vormen op de nu aangenomen regel 'ge' = gɣ' nader bekijken dan blijkt het volgende:
- de combinatie GENT kan nooit de combinatie 'gɣ' bevatten;
- de combinatie GEMZ evenmin;
- de combinatie GENR evenmin.

LETTER	CIJFERCODE
b f p v	1
c g j k s x z	2
d t	3
l	4
m n	5
r	6

Met dit systeem heeft Tom Munnecke een BASIC (Microsoft Basic) programma geconstrueerd. Het overkwam Munnecke nogal eens dat zijn wat ongewone naam verkeerd werd geschreven. Zijn SOUNDEX programma gaf de volgende coderingen op de door hem verzamelde foutieve spellingen.

NAAM	CODE
Munnecke	M520
Minnecke	M520
Munnuke	M520
Munneake	M520
Munneeke	M520
Munickey	M520
Muneeck	M520
Monkey	M520
Muneick	M520
Munnick	M520
Monnecks	M521
Muuncake	M522
Munnedie	M530
Lunnecke	L520
Munnecle	M524

Van de 16 varianten kwamen er dus 10 op hun plaats terecht. Toch is het systeem nog aardig grof. Zouden de namen alleen verschillen op de middenletters G en Z (in het Nederlands bijvoorbeeld lezen en legen) dan zouden ze worden samengetrokken tot een enkele naam. Het gevolg is dan ook dat gebruik van SOUNDEX toch nog veel handwerk geeft. Moet men met fijnere mazen werken, dan is een meer op klankwetten gebaseerd systeem aan te bevelen.

- Het programma KLANKBAS

Het programma KLANKBAS werd ontwikkeld voor het Duits. Het wordt vooral gebruikt voor spellingsvarianten in archiefonderzoek voor economisch-historisch onderzoek. Toepassing van SOUNDEX op Duitse namen bleek namelijk te veel fouten op te leveren. Het is duidelijk dat alleen de transpositietabel van belang is voor dit karwei. Ook is duide-

lijk dat de transpositietabel per soort karwei verschillend kan zijn. Voor het Duitse onderzoek bleek het best een puur taalkundige transpositietabel te voldoen.

1. Wat kan wegvallen

Zoals al bij SOUNDEX bleek zijn klinkers niet erg belangrijk bij dit soort werk. Ze kunnen gemist worden. Voor het Duits kwam daar nog bij dat ook de letter 'j' en de letter 'h' niet bijdroegen tot onderscheid tussen twee echt verschillende woorden/namen. De eerste tabel is dus de vocaaltabel bestaande uit: 'a', 'e', 'i', 'o', 'u', 'j', 'h'.

2. Stemhebbend/stemloos

Het Duits bevat een serie medeklinkers die eigenlijk elkaars gelijke zijn. Ze verschillen slechts op één punt, namelijk de een wordt uitgesproken met gebruik van de stembanden, de ander zonder (zie in het Nederlands de paren 'd'/'t' en 'b'/'p'). De omzetting van stemhebbend naar stemloos geeft de volgende transpositietabel:

b → p	g → k
d → t	c → k
v → f	z → s

Opmerking. De c hoort niet echt in dit rijtje thuis, omdat de c niet als iets anders wordt uitgesproken dan als k of ts.

3. Dubbele letters

Alle dubbele letters die voorkomen na de transposities worden enkel gemaakt.

4. Speciale combinaties

De 'tz' wordt tot 's' herschreven, want 'tz' is gelijk 'z'!
De 'ph' wordt herschreven tot 'f'.

Met deze eenvoudige transposities bleek het probleem van de tientallen variante spellingen in het Duits te zijn opgelost.

• De METEOR regels

De METEOR regels zijn eenvoudig te schrijven.

1. Wegvallen

```
(*    ( A $ )      ( 2 )    /)
(*    ( E $ )      ( 2 )    /)
(*    ( I $ )      ( 2 )    /)
(*    ( O $ )      ( 2 )    /)
(*    ( U $ )      ( 2 )    /)
```

Daarmee is het probleem zoals het hier werd gesteld opgelost.
Alleen de METEOR regels zijn nog nodig en wel als volgt:

```
(* ( G E N T )          VR )
(* ( G E M Z )          VR )
(* ( G E N R )          VR )
(* ( G E ($.1)  )       ( 1 ϒ 3 )     *)
(VR (     )             (       )     *)
```

Verdere problemen en regels die in de beschrijving van het probleem der leesmachine aan de orde werden gesteld zijn de volgende:

1. Stemhebbende medeklinkers worden stemloos aan het eind van een woord.
2. Dubbele medeklinkers worden enkel.
3. Het patroon 'ng' aan het eind van een woord wordt 'n1'.
4. Het patroon 'ig' kan deel zijn van 'uig'.
5. Moeilijke woorden: tafel/tabel/beregelen/evenredig/berevel.

De regels in METEOR daarvoor zijn:

1. Van stemhebbend naar stemloos voor b en d:

```
(*  ( B ( $.0 )  )     ( P )       VR)
(*  ( D ( $.0 )  )     ( T )       VR)
(VR   (     )          (   )  .......
```

2. Dubbele medeklinker wordt enkele:

```
(*  ( B B )     ( 1 )       /)
```

OPGAVE. Schrijf de regels voor de andere dubbele medeklinkers.

3. 'ng' aan het eind van een woord:

```
(*  ( N G ( $.0 )  )     ( N1 )     *)
```

VRAAG. Waarom wordt in de rechterhelft N1 aan elkaar geschreven?

4. Het patroon '-ig' kan behoren tot 'uig':

```
(*  ( U I )     ( 6 )       /)
(*  ( I G )     ( ϒ 2 )     /)
```

5. Moeilijke woorden:

```
(*  ( A F E )        ( 1 2 ϒ )            *)
(*  ( A B E )                             VR)
(*  ( B E R E V )    ( 1 2 2 3 ϒ 5 )      *)
(*  ( B E )          ( 1 ϒ )              *)
```

```
(*  ( R E G E )     ( 1 2 2 G γ )        *)
(*  ( L E N )       ( 1 γ )              *)
(*  ( E V E N )     ( 1 1 2 γ 4 )        *)
(*  ( R E D I G )   ( 1 2 2 3 γ X )      *)
```

OPGAVE. Vul het programma voor 'ge' aan met de regels voor de woorden: gevel, generaal, gesel.

3.3.1.2 Schrijfvarianten/schrijffouten

Een van de zeer grote onaangenaamheden van de computer, dus ook van de tekstmachine, is dat hij strikt formeel en schoolmeesterachtig is. Daardoor krijgt men triomfantelijke antwoorden als: NO SUCH COMMAND, bijvoorbeeld wanneer de programmeur schrijft RETUNR in plaats van RETURN. Schrijft men op een record voor zijn administratie in plaats van de gezochte meneer Smith bij vergissing Smyth of Smit dan zal niemand de gezochte gegevens krijgen. Deze verschrijvingen zijn dus in feite rampzalig. De mogelijkheid bestaat namelijk ook dat de triomf van de machine in termen van NO SUCH COMMAND uitblijft en dat er ongemerkt foutieve resultaten uit de bus komen. Vooral geschiedkundigen liepen bij hun onderzoek in volkstellingen of oude archieven tegen het feit aan dat de computer te letterlijk is en weigert te begrijpen wat er bedoeld wordt. Er zijn verschillende pogingen ondernomen om langs dit ongemak heen te komen.

- SOUNDEX

Een beroemd computerprogramma waarmee men het probleem van de spellingsvarianten of schrijffouten kan vermijden, althans gedeeltelijk, is het programma SOUNDEX. Het werd ontwikkeld bij een groot onderzoek in belastingbestanden uit 1890 in het kader van een groot socio-economisch-historisch onderzoek. De techniek waarop SOUNDEX berust is eenvoudig. Iedere naam krijgt een code van vier symbolen. De eerste daarvan is de eerste letter van de naam. De drie andere symbolen zijn getallen die de klanken representeren die in de rest van de naam voorkomen. Op die manier komen alle namen die gelijk klinken bij elkaar te staan. Hoe men precies een SOUNDEX code moet construeren kan men vinden in Knuth's 'The Art of Computer Programming deel 3'.

Wat alle SOUNDEXen gemeen hebben (er zijn verschillende varianten van) is dat de klinkers worden weggelaten. De andere letters krijgen een cijfercode, bijvoorbeeld:

```
(*     ( J $ )          ( 2 )     /)
(*     ( H $ )          ( 2 )     /)
```

2. Stemhebbend/stemloos

```
(*     ( B )            ( P )     /)
```

OPGAVE. Schrijf de andere transposities van deze categorie.

3. Dubbele letters

```
(*     ( P P )          ( 1 )     /)
```

VRAAG. Hoeveel regels zijn hiervoor maximaal nodig?

4. Speciale combinaties

```
(*     ( T Z )          ( S )     /)
(*     ( P H )          ( F )     /)
```

VRAAG. Waar in het programma moeten deze twee regels komen te staan?

3.3.2 Grammatica's: Russische poppen, cacao en ketters

Er is wel eens gezegd, door S. Gill in 1960 al, dat "als er computers in de Middeleeuwen bestaan hadden, dan zouden er programmeurs op de brandstapel omgebracht zijn wegens ketterij". Het is bijna zeker dat een van de voornaamste ketterijen het (on)geloof in recursie zou zijn. De (on)deugden van recursie als programmeertechniek staan al jaren ter discussie. Maar wat is het dan?

- Wat is recursie?

Het begrip recursie zit in allerlei zaken: verhalen in een verhaal, een film over een film, afbeeldingen binnen een afbeelding (denk bij dit laatste aan Droste's cacaoblikken), Russische poppen binnen Russische poppen. In de programmeerwereld wordt recursie toegepast met behulp van het recursief vastleggen van een programma. Dit heet ook wel recursieve definitie.

De oppervlakkige waarnemer krijgt over dit soort programma's de indruk dat zij gedefinieerd worden in termen van zichzelf: zij lijken dus nooit te kunnen stoppen met executie indien ze eenmaal in gang gezet zijn. Indien recursieve definitie correct gebeurt is er echter nooit sprake van die oneindige executie. Dat komt doordat een recursieve definitie nooit iets letterlijk in termen van zichzelf vastlegt, maar altijd definieert in termen van eenvoudiger versies van zichzelf.

Wat hiermee bedoeld wordt komt later aan de hand van praktijkvoorbeelden aan de orde.

Een van de meest voorkomende recursies is een lopende taak uitstellen ten gunste van een eenvoudiger taak. Een voorbeeld is iemand die telefoneert met A, als B belt. Tegen A zegt hij: "Hebt u een ogenblikje?" en drukt de gesprekstknop in, waardoor hij met B verbonden is. Nu belt C en dezelfde behandeling overkomt B. Dit kan zo doorgaan.

Maar laten we voor het gemak zeggen dat de recursie stopt bij C. Dan komt onze 'telefonist' weer tevoorschijn bij B (in het Engels: 'pops' back up to B) en vervolgt zijn gesprek met B. Ondertussen trommelt A met zijn vingers op zijn stoel om zijn verveling te doden.. ... Het eenvoudigste geval nu doet zich voor als het getelefoneer met B afgemaakt wordt en uiteindelijk A weer aan de beurt komt. Het kan echter voorkomen dat het gesprek met B weer verder gaat en er een nieuw gesprek, D, binnenkomt. B wordt weer weggedrukt naar de wachtlijst (in het Engels: pushed into the stack of waiting callers) en er wordt naar D geluisterd. Nadat D afgehandeld is, terug naar B en tenslotte terug naar A. Deze 'telefonist' is nogal robotachtig, maar we illustreren recursie hiermee zeer nauwkeurig.

• De borden in een cafetaria

Het mechanisme van recursie en enkele (Engelse) termen die daarvoor gebezigd worden, zijn genoemd in de vorige paragraaf. Dit zijn namelijk de termen push, pop en stack (stapel) (ook wel push-down stapel genaamd), die geïntroduceerd werden als onderdeel van IPL, een van de eerste programmeertalen voor Kunstmatige Intelligentie. De term 'push' betekent dat je het werken aan je huidige taak uitstelt, zonder overigens te vergeten dat je ermee bezig was, en je begint een andere taak. Deze nieuwe taak wordt gewoonlijk 'op lager niveau' dan de vorige taak genoemd. De term 'pop' geeft het omgekeerde weer, namelijk dat het werk op een niveau afgesloten wordt en dat voortgegaan wordt precies daar waar je een niveau hoger gebleven was.

Maar hoe weet je precies waar je gebleven was? Het antwoord is dat je de daarvoor relevante informatie op een 'stapel' opbergt. Een stapel is dus een tabel die je meedeelt (1) waar je was in iedere niet beëindigde taak (vaktaal: het 'terugkeeradres') en (2) wat de relevante gegevens waren op het moment van onderbreking (vaktaal: de 'veranderlijke betrekkingen', in het Engels: 'variable bindings'). Als je tevoorschijn komt om een taak te vervolgen (to pop back up), zorgt de stapel voor je informatie, zodat je je niet verloren voelt. In het voorbeeld van de telefoon vertelt de stapel je wie er wacht op ieder niveau en waarover je bij iedere onderbreking sprak.

De termen 'push', 'pop' en 'stapel' komen uit de cafetariawereld. Zij worden daar gebruikt voor de stapel borden waar een veer onder zit. Als je er een bord bovenop plaatst (= push) blijft de stapel ogenschijnlijk even hoog en als je er een bord vanaf neemt komt de stapel met een bord omhoog (= pop).

• Recursie bij talen

De taalkundige, vooral de transformationeel-generatieve richting in de taalkunde, veronderstelt bij de mens een grammatica die gebruik maakt van push-down stapels, alhoewel de moeilijkheid om een zin te begrijpen sterk toeneemt met het aantal borden op de stapel. Het spreekwoordelijke Duitse verschijnsel van het 'werkwoord aan het eind' bij verstrooide professoren, die een zin beginnen, een college lang doorpraten en dan eindigen met een serie werkwoorden, waardoor hun gehoor, wier stapel allang de feiten niet meer kon bevatten, totaal verloren is, is een voorbeeld van push en pop in de taalkunde.

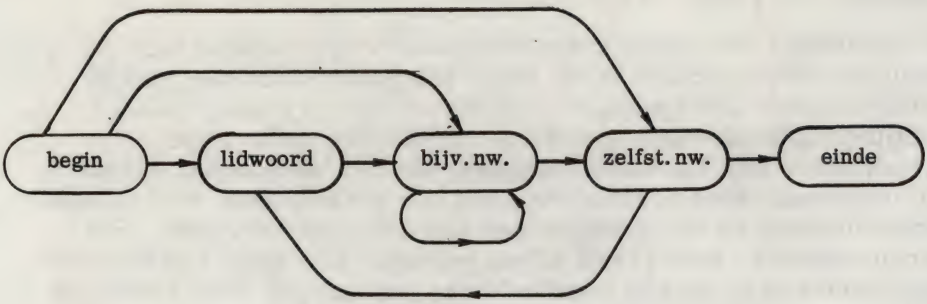
In de praktijk formuleert men zo, dat dat verschijnsel niet voorkomt, dat wil zeggen dat de hoogte (diepte) van de stapel minimaal is. Op school hebben we de grammatica van het Nederlands geleerd. Een van de uitgangspunten van die grammatica was dat Nederlands bestaat uit zinnen. Zinnen kunnen bovendien weer bestaan uit zinnen in zinnen. Dat is de reden dat men geprobeerd heeft een recursieve definitie voor het Nederlands te vinden.

• Recursieve overgangsnetwerken

De veronderstelde grammaticale gedaante van zinnetjes geeft ons een interessante gelegenheid om recursieve processen en structuren te beschrijven, in de gedaante van zogeheten Recursieve Overgangsnetwerken (in het Engels: Recursive Transition Networks (= RTN)).

Een RTN is een diagram met diverse paden die gevolgd kunnen worden om een zekere taak te volvoeren. Ieder pad bestaat uit een aantal knooppunten, getekend als rondjes of als rechthoeken, die verbonden zijn door lijnen of door pijlen. De naam van een RTN staat meestal naast of onder het diagram; deze naam beschrijft in het kort de hoofdtaak van zo'n RTN. Bijvoorbeeld: voor het herkennen van zelfstandige naamwoorden heet een RTN: NP. De taak van een RTN zoals omschreven in het bijbehorende diagram begint in een knooppunt met een of andere aanduiding voor start of begin; soms is het een driehoekig symbool. Het einde van de opdracht valt samen met een knooppunt dat 'einde' heet. Soms is dat ook een driehoek, maar het kan ook een cirkel zijn met daarin concentrisch een kleinere cirkel. Alle andere knooppunten bevatten òf erg korte precieze omschrijvingen voor taken òf namen van andere RTN's, die daar dus als taak opgeroepen worden. Iedere keer dat een knooppunt bereikt wordt, dienen eerst de taken die daarin genoemd worden te worden uitgevoerd. Als dus in een knooppunt een andere RTN vermeld wordt, springt men naar de desbetreffende RTN om deze uit te voeren.

Als voorbeeld nemen we een eenvoudige versie van een RTN, om een zelfstandig-naamwoord-groep in een Nederlandse zin te vinden. Een zelfstandig-naamwoord-groep is een serie bij elkaar horende woorden, waarvan er één een zelfstandig naamwoord is, bijvoorbeeld 'de oude man'. Deze groep wordt meestal aangeduid als nominale frase (Engels: Noun Phrase = NP).



Figuur NP

Links begonnen in de figuur NP blijkt, dat een NP kan beginnen met een lidwoord, waarachter gevoegd een bijvoeglijk naamwoord zowel als een zelfstandig naamwoord. Ook is het volgens deze RTN correct Nederlands om te starten met een bijvoeglijk naamwoord, òf om te beginnen en te eindigen met slechts een zelfstandig naamwoord.

Het bijvoeglijk naamwoord heeft een pijl naar zichzelf; dit kan willekeurig vaak voorkomen, bijvoorbeeld: gekke, lieve, dwaze, flinke man. Een volledige opsomming van alle door de figuur NP toegelaten naamwoordelijke gedeelten geven we hierna, met inachtneming van de afspraken dat:

- verplichte volgorde aangegeven wordt met het vermenigvuldigingsteken '.'
- tegelijkertijd toelaatbare volgorden aangegeven worden met het optellingsteken '+'
- herhaald voorkomen van een symbool aangegeven wordt met een ster, dus '*'
- er afkortingen gebruikt worden: DET voor lidwoord, ADJ voor bijvoeglijk naamwoord en ZN voor zelfstandig naamwoord.

Het diagram kunnen we dan als volgt omschrijven:

NP = N
NP = ADJ*.N
NP = DET.N
NP = DET.ADJ*.N

Deze formulering kan nog korter door de afspraak over gelijktijdig toelaten van mogelijkheden te formaliseren met het plusteken; er komt dan:

NP = N + ADJ*.N +
+ DET.N + DET.ADJ*.N

Als we het symbool 1 invoeren voor de niet-waarneembare taaluiting komt er nog korter:

NP = [1+ADJ* + DET.(1+ADJ*)].N

Indien nu rechts van het = teken het symbool NP ook zou voorkomen heet deze naamwoordelijke groep recursief. In het diagram van het RTN betekent dat het voorkomen van een knooppunt met de naam van dit RTN erin! De hier zojuist ingevoerde algabraïsche notatie maakt het dan mogelijk alle toelaatbare zinnen volgens zo'n RTN van te voren te voorspellen. Voorwaar een luisterrijke zaak!

Voor meer inzicht in deze wiskundige aanpak van taal zie de boeken van Conway, Ginsburg, Hennie, Hunt, Marcus en Starke in de literatuurlijst.

Hiermee zijn we terug bij de paragraaf over SNOBOL, waarin we een uitgebreider voorbeeld van een zinnenprogramma gaven. Grondbegrippen voor computergrammatica's zijn recursiviteit en netwerken. Voor we verder gaan nu echter eerst een programma.

3.3.2.1 Voor de stille uurtjes: zinnen genereren

Eenzaamheid is volgens advertenties in Amerikaanse bladen op te lossen met briefwisseling. Er worden in dat land standaard brieven voor relatiebemiddeling te koop aangeboden. Wij stellen ons voor die brieven met de computer te genereren in eenzame uren. Voor nog vreemdere toepassingen zie het boek van Hofstädter. We beginnen niet met een literaire liefdesbrief, maar met eenvoudige zinnen zoals:

Jan bemerkte Jenny
Henk zag Mia
Martin groette Beatrijs

Deze zinnen zijn van het type: zelfstandig naamwoord, werkwoord en dan weer een zelfstandig naamwoord (Engels: noun phrase, verb, noun phrase) en afgekort mag natuurlijk ieder zelfstandig naamwoord en ieder werkwoord op de desbetreffende plaatsen ingevuld worden. Dit 'invullen' heet in taalkundige kringen ook wel 'herschrijven'.

Het uiteindelijke resultaat van dit herschrijven noemt men wel de uiteindelijke uitdrukking (Engels: terminal expression). De niet-uiteindelijke uitdrukking (Engels: non-terminal expression) kan dus herschreven worden. De meest abstracte non-terminal is in dit verband het symbool S voor de gehele zin (Engels: sentence). De S wordt in dit eenvoudige voorbeeld dus herschreven tot de non-terminal symbolen NP, VB, NP en de grammatica daarvoor luidt dan:

```
(SETQ GRAM '((S ((NP, VB, NP)))
             (NP (JAN, JENNY, HENK, MIA, MARTIN, BEATRIJS))
             (VB (BEMERKTE, ZAG, GROETTE))))
```

Er is dus slechts een wijze voor het herschrijven van het symbool S, dat wil zeggen er is slechts een zinsstructuur toegelaten. De symbolen NP en VB kunnen op meerdere manieren in terminal symbolen eindigen. Om deze meerdere keuzen door de computer te laten doen, plaatsen we de lijst van herschrijfkeuzen voor een non-terminal symbool als de eigenschap (Engels: property) van de indicator KEUZEN en we berekenen onmiddellijk het aantal mogelijke KEUZEN, die op zijn beurt weer de property is van de indicator GETALKEUZE.

Nu kunnen we de toepassing van voorgaande grammatica als volgt programmeren:

```
(DEF (TOEP (GRAMMATIREGEL) (PROG( )
        (PUTPROP (CAR GRAMMATIREGEL) 'KEUZEN
                 (CADR GRAMMATIREGEL))
        (PUTPROP (CAR GRAMMATIREGEL) 'GETALKEUZE
                 (LENGTH (CADR GRAMMATIREGEL))))))
```

De operator PUTPROP legt de proporties vast. Deze functie moet op iedere grammaticaregel toegepast worden en daartoe gebruiken we de MAPC, die regel voor regel afwerkt:

```
(MAPC GRAM 'TOEP)
```

Ieder woord dat nog KEUZEN heeft is nu non-terminal en indien die er niet meer zijn, nemen we dat woord zelf voor terminal symbool.

Nu kunnen we de zinnetjes-generator construeren: laten we beginnen met de zin (S). S wordt vervangen door een van zijn KEUZEN. In dit geval is er maar een, dus komt er (NP, VB, NP). De eerste NP wordt vervolgens vervangen door een van de KEUZEN door gebruik te maken van een random generator met als argument het aantal (= de lengte) woorden in de desbetreffende NP-groep (dat getal is te vinden in GETALKEUZE).

Neem voor het gemak even aan dat de RANDOM generator het woord HENK aanwijst, dan wordt de zin (HENK, VB, NP). Omdat HENK geen KEUZEN heeft gaat het programma naar het werkwoordsdeel VB dat vervangen wordt door een van zijn KEUZEN, bijvoorbeeld BEMERKTE. De zin S wordt nu herschreven tot (HENK BEMERKTE NP). BEMERKTE heeft geen KEUZEN, zodat vervolgd wordt met NP, deze wordt op zijn beurt vervangen door een van de KEUZEN; we nemen als voorbeeld MIA, dan wordt de uiteindelijke formulering (HENK BEMERKTE MIA). MIA heeft verder geen KEUZEN en er zijn geen woorden meer te verwerken, dus stopt de zinnengenerator.

```
(DEF (ZINGEN () (PROG (EIND TIJDEL KEUZEN)
        (SETQ EIND (SETQ TIJDEL (COPY '(S))))
 LUS    (COND ((NULL TIJDEL) (RETURN EIND))
              ((SETQ KEUZEN (GET (CAR TIJDEL) 'KEUZEN))
               (REPLACE TIJDEL (CAR(VINDNDECDR (SUB
                  (RANDOM (GET (CAR TIJDEL) 'GETALKEUZE)))
                     KEUZEN))))
               (T (SETQ TIJDEL (CDR TIJDEL))))
        (GO LUS))))
```

De zingeneratie wordt begonnen als (S) van het Engelse sentence, en naar de oorspronkelijke geheugenplaats voor (S) wordt verwezen door EINDE. TIJDEL verwijst naar de geheugenplaats waaraan op dat moment gewerkt wordt. Als de eerste (= CAR) van die plaats terminal is, dat wil zeggen dat hij geen KEUZEN heeft, wordt TIJDEL voortgeschoven op de lijst. Anders wordt (CAR TIJDEL) vervangen (= REPLACE) door een willekeurige (= RANDOM) KEUZE en blijft TIJDEL onveranderd, maar zal de CAR ervan meestal wel gewijzigd worden.

REPLACE verandert het voorlopige EINDresultaat door de juiste RPLACA's en RPLACED's te doen. Een hulpfunctie LAATSTE vindt het laatste stuk van de te doorzoeken lijst.

```
(DEF (REPLACE (PLAATS UITDR) (COND
              ((ATOM UITDR) (RPLACA PLAATS UITDR))
              ((RPLACA PLAATS (CAR SETQ
                                 UITDR (COPY UITDR))))
                 (PROG2(RPLACD (LAATSTE UITDR)
                               (CDR PLAATS))
                       (RPLACD PLAATS (CDR UITDR))))
)))
```

Let op dat, indien UITDRukking een lijst is, een copie gemaakt wordt om de geheugenstructuur van UITDRukking te behouden. Overigens is de functie LAATSTE in sommige systemen aanwezig onder de naam FF:

```
(DEF (LAATSTE (LIJST) (COND
        ((NULL (CDR LIJST)) LIJST)
        (T (LAATSTE (CDR LIJST))) )))
```

De aanroep: RANDOM in de ZINGENerator is in elk LISP systeem aanwezig en dient om een willekeurig getal tussen 0 en 1 op te leveren. De hier gevraagde functie moet een willekeurig element uit KEUZEN opleveren. Voor een goede uitleg van de ideeën betreffende generatie van RANDOM getallen, zie Knuth, deel 2.

OPGAVEN

1. Definieer de in de voorgaande tekst benodigde RANDOM functie. Een voorbeeld voor zo'n functie vormt de volgende WILLEKEUR, die uit de lijst X een willekeurig element kiest op grond van een random getrokken getal dat in Y opgegeven wordt.
   Een aanroep van WILLEKEUR zou luiden:

   ```
   (WILLEKEUR '(A, B, C) (RANDOM Ø))
   ```

   indien het LISP systeem voor (RANDOM Ø) een getal tussen Ø en 1 oplevert. Voor meer informatie hierover zie Knuth, deel 2.
   Een definitie van WILLEKEUR zou kunnen luiden, met de *operator als symbool voor het vermenigvuldigingsteken:

   ```
   (DEF (WILLEKEUR (X, Y) (COND
            ((< (* (LENGTH X) Y) 1) (CAR X))
            ((< (* (LENGTH X) Y) 2) (CADR X))
            ((< (* (LENGTH X) Y) 3) (CADDR X))
   ```

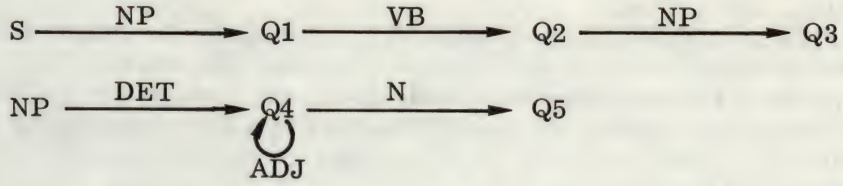
2. Draai de ZINGENerator met behulp van onderstaande grammatica:

   ```
   ((S    (NP VB NP))
    (NP   (DET ADJ N))
    (VB   (at liefkoosde rook))
    (DET  (de een het))
    (ADJ  (arme leuke domme))
    (N    (programmeur muizenbiefstuk)))
   ```

3. Schrijf een programma dat als MONITOR fungeert voor deze ZINGENerator, dat wil zeggen een functie die een grammatica inleest en vervolgens zorgt dat alles automatisch verloopt.
   Zie hieromtrent het boek van Siklossy (zie literatuurlijst).
4. Definieer een functie LAATSTE van een lijst.
5. Definieer een functie OPEENNALAATSTE van een lijst.
6. Definieer een functie die de N-de CDR vindt: VINDNDECDR.
7. Waarom is COPY belangrijk in voorgaande ZINGENerator?

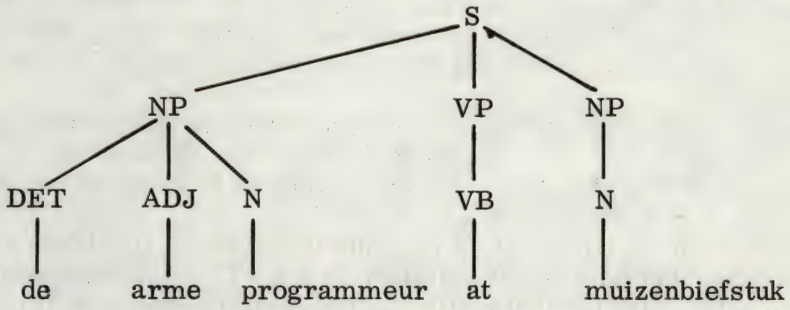
3.3.2.2 RTN en ATN

In een enkele oogopslag kan men een beeld geven van wat een eenvoudige RTN is. Als volgt:



Woordenboek (de DET)
(arme ADJ)
(programmeur N)
(at VB)
(muizenbiefstuk N)

De structuurboom van deze zin ziet er als volgt uit. (Uit deze boom krijgt men ongeveer een indruk waar gepusht en gepopt wordt.)



Het gemak van een RTN is de doorzichtige wijze van noteren. Bezwaren zijn er verschillende. Ten eerste kan het zijn dat er meer dan één mogelijkheid is bij het klimmen van boven naar beneden door de boom en terug. Een beroemde voorbeeldzin hiervan is:

> Als vliegen achter vliegen vliegen dan vliegen vliegen achter vliegen

Op deze zin komen we later terug, wanneer het erom gaat, woordsoorten automatisch door de computer te laten benoemen. Nu echter dient hij slechts om aan te tonen hoe zwak het RTN mechanisme van de contextvrije grammatica eigenlijk is. 'Vliegen' kan een zelfstandig naamwoord zijn, maar ook een werkwoord. In het woordenboek voor een RTN moeten beide mogelijkheden zijn opgenomen. De computer kiest er willekeurig een van voor zijn analyse en loopt hoogstwaarschijnlijk vast. Dan moet hij terug naar het punt van keuze en een

andere keuze proberen. In deze zin alleen al zijn zes punten van keuze. Dus de mogelijkheid bestaat dat de computer een zestal keuzes moet gaan verifiëren voor hij de juiste gevonden heeft. Dit procédé van telkens terugkeren en opnieuw starten vanuit een bepaald punt noemt men back tracking.

De meerduidigheid van woorden in natuurlijke taal maakt het werken met een RTN in feite onmogelijk vanwege de enorme back tracking die telkens weer nodig is bij een contextvrije grammatica die werkt met alleen een woordenboek waarin alle theoretische mogelijkheden staan genoteerd. Ook Chomsky kwam er dus al vrij snel achter dat ContextVrije (CF) grammatica's niet geschikt zijn voor de beschrijving van natuurlijke taal. Hij bedacht daarop zijn transformationeel-generatieve grammatica. Met deze grammatica kan men ook feiten als vraagzinnen en de lijdende vorm beschrijven, wat in een CF grammatica al niet eens mogelijk was. De netwerk-notatie – men wilde die natuurlijk niet zomaar opgeven, hij was immers helder en doorzichtig – moest dan wel worden uitgebreid, er moesten meer mogelijkheden van bijschrijven langs de lijntjes komen. William Woods bedacht dat allemaal en hij noemde zijn nu uitgebreide RTN-formalisme ATN, voor Augmented Transition Network. In het woordenboek kan nu ook anderssoortige informatie worden opgenomen. Zo kan er bijvoorbeeld inkomen dat 'paarden' een zelfstandig naamwoord in het meervoud is of dat 'was' de verleden tijd van een werkwoord is.

Verder is de ATN niet meer alleen afhankelijk van het pad waarlangs de analyse verloopt, omdat in de registers stukken van de analyse kunnen worden bewaard. Zo kan onthouden worden dat een NP onderwerp van de zin was of iets dergelijks.

In feite zijn de lijnen uit de RTN die de precieze voortgang van de machine door het netwerk beschrijven in het ATN vervangen door programma's, die flexibeler zijn dan het rechttoe-rechtaan lijntje. Het zwakke punt hier is wel weer het woordenboek. De informatie die daarin zit ligt vast en de ATN is er volledig van afhankelijk. Dat wil dus in feite zeggen dat het hele analysewerk wordt gedaan door de taalkundige of programmeur wanneer hij de informatie in het woordenboek stopt. Dat betekent dat we op dezelfde problemen uitkomen als bij de vliegende vliegenzin. Ook de ATN dient alle mogelijkheden te gaan uitproberen wanneer hij vastloopt; de valkuil van ATN is de back tracking.

Een verder bezwaar van ATN's is, dat alle aandacht ligt bij de zinsontleding zoals bij de taalkundige theorie, die werd bedacht door Chomsky c. s. Echte ruimte voor wat woorden betekenen, dat wil zeggen een echte plaats voor wat de mens van zijn taal begrijpt, is er niet in de ATN's.

• Een ATN programmeren in LISP

De bedoeling van een transitie-netwerk (dit omvat dus RTN zowel als ATN) is het vasthouden van informatie over een grammatica. Als voor de overgangen van de ene toestand naar de andere in een RTN extra informatie nodig is, noemt men het netwerk 'Augmented Transition Network' (= uitgebreid overdrachtsnetwerk). Bijvoorbeeld: bij de lidwoorden 'de', 'het' en 'een', aangeduid met de non-terminal LIDW, kan eerst bekeken worden of het een bepaald of een onbepaald lidwoord betreft. Ook kan extra informatie ingewonnen worden of een zelfstandig naamwoord enkelvoud of meervoud is. Nu zal het de computer niet interesseren wat voor soort informatie erbij gehouden wordt tijdens de analyse van een zin. De 'extra informatie' mag dus ook andere zaken betreffen dan het onthouden van de woordklassen, zoals gezegd. Bij de ATN's gaat het in feite om extra boekhouding over de verrichte analyse, waarbij die boekhouding opgehangen wordt aan de overgangen van de ene naar de andere toestand in het netwerk.

Een goede en eenvoudige beschrijving van ATN's wordt gegeven door Winston in zijn boek over Kunstmatige Intelligentie (zie literatuurlijst), waarbij hij het mechanisme voor b a c k t r a c k i n g om didactische redenen achterwege laat. Wij volgen hier zijn methode van uitleggen. Deze methode heeft het nadeel, dat er vanuit een toestand in een netwerk geen alternatieven vanuit dezelfde toestand bereikt kunnen worden, immers, er is geen back tracking opgenomen.

Iedere toestand in het netwerk is verbonden met een PROGram tekst, hierna in de vorm van CONDitional statements. S is de zin, dus S is de variabele waarvan de waarde de lijst van nog te analyseren woorden is en W staat voor het onderhanden woord, de CAR van S dus. Nu kijkt het volgende LISP programma naar een lidwoord, gevolgd door een onbepaald aantal bijvoeglijke naamwoorden, die op hun beurt weer gevolgd worden door een zelfstandig naamwoord, dat zijn de toestanden Q3, Q4 en Q5 in de RTN over de muizenbiefstuk, zoals hiervoor gegeven.

```
Q3 (COND ((MEMBER 'LIDW (GET W 'KENMERKEN))
          (SETQ W (CAR (SETQ S (CDR S))))
          (GO Q4))
         (T (RETURN NIL)))
Q4 (COND ((MEMBER 'BIJVNW (GET W 'KENMERKEN))
          (SETQ W (CAR (SETQ S (CDR S))))
          (GO Q4))
         (T RETURN NIL)))
Q5 (COND ((MEMBER 'ZELFSTNW (GET W 'KENMERKEN))
          (SETQ W (CAR (SETQ S (CDR S))))
          (RETURN T))
         (T (RETURN NIL)))
```

De bepaling van een woordklasse verschijnt dus op de property list onder de eigenschappennaam KENMERKEN. Andere kenmerken, zoals ENK voor enkelvoud of BEV voor de bevelende wijs kunnen op dezelfde property list opgeborgen worden.

In bovenstaand LISP programma wordt duidelijk dat de handelingen voor het vinden van kenmerken, het toevoegen van nieuwe kenmerken en het veranderen van de inhoud van S en W telkens opnieuw geprogrammeerd moeten worden bij iedere toestand Q. Daarom maken we een aantal functies die deze taken standaardiseren en die dus telkens slechts aangeroepen behoeven te worden om het werk over te nemen. We maken er een drietal. De eerste SLIKt iedere zin S in om te zien of de inhoud van die S toevallig leeg is; zo ja dan zet hij de inhoud van W op leeg (= NIL):

```
(DEF (SLIK () (COND
        (S (SETQ S (CDR S))(SETQ W (CAR S)))
        (T (SETQ W NIL)))))
```

De volgende functie vereenvoudigt de aanroep
(GET W 'KENMERKEN) in (GETK W):

```
(DEF (GETK (W)(GET W 'KENMERKEN)))
```

Tenslotte maken we alvast een functie PUTK die aan ieder onderhanden woord W de nieuwe kenmerken NK toevoegt, waarbij eerst getest wordt of de lijst van nieuwe kenmerken NK soms leeg is; dan volgt de waarde NIL voor PUTK. Vervolgens kijkt de functie PUTK bij de nieuwe kenmerken of deze soms slechts uit een enkel atoom bestaan: (ATOM NK) en zo ja, dan wordt de eerste keer dat NK in de oude KENMERKENlijst voorkomt geveegd (= EFFACE), en het nieuwe kenmerk geplakt (= CONS) vooraan op de zojuist geschoonde KENMERKENlijst:

```
(DEF (PUTK (W NK)(COND
        ((NULL NK) NIL)
        ((ATOM NK)(PUT W
                (CONS NK (EFFACE NK (GETK W)))
                'KENMERKEN))
        (T (PUT W(UNION NK (GETK W))
            'KENMERKEN)))))
```

De vreugde van het vereenvoudigen levert nu voor de toestanden Q3, Q4 en Q5 van de RTN:

```
Q3 (COND ((MEMBER 'LIDW (GETK W))
          (SLIK)
          (GO Q4))
          (T (RETURN NIL)))
Q4 (COND ((MEMBER 'LIDW (GETK W))
          (SLIK)
          (GO Q5))
          (T (RETURN NIL)))
Q5 (COND ((MEMBER 'ZELFSTNW (GETK W))
          (SLIK)
          (RETURN T))
         (T (RETURN NIL)))
```

Deze drie toestanden met daaraan verbonden PROGram-fragment vormen slechts een onderdeel van een eenvoudige RTN. Het is aardig om te zien hoe het grotere geheel eruit ziet. Daar is nogal wat voor nodig:

- Voor iedere woordklasse is er een analysefunctie.
  De functie voor zelfstandige naamwoorden noemen we ZNW.
- Voor iedere nieuw ontdekte woordgroep genereren we een nieuw atoom. Bijvoorbeeld: de derde zelfstandige naamwoordgroep heet ZNW3. Dit nieuwe atoom heeft dan de property list met KENMERKEN die informatie bevatten over de desbetreffende groep.
- Het verband tussen iedere ZNW-groep en de groep waartoe deze behoort, dient bekend te zijn. Dit wordt bereikt door de naam van de overkoepelende groep aan de specialistfunctie als argument door te geven en door de standaardfunctie PUT te gebruiken om desbetreffende afkomst en onderdelen vast te leggen en te kunnen vereenvoudigen.
- Als het zoekwerk naar een groep ergens vastloopt, zet dan S en W terug op hun inhoud voordat de stagnatie optreedt.

Deze ideeën zijn vervat in de volgende functie ZNW, die in plaats van (RETURN T) en (RETURN NIL) uit voorgaand programmafragment de instructies (GO WIN) en (GO LOSE) bevat:

```
(DEF (ZNW (AFKOMST KENMERKEN)(PROG (KNOOP HOLD)
                  (SETQ HOLD S)
                  (SETQ W (CAR S))
                  (SETQ NODE (GENNAME 'ZNW))
                  (PUTK NODE KENMERKEN)
        Q3        .
                  .
                  .
```

```
WIN     (PUT NODE AFKOMST 'AFKOMST)
        (PUT AFKOMST
             (CONS NODE
                   (GET AFKOMST 'ONDERDEEL))
             'ONDERDEEL)
        (RETURN NODE)
LOSE    (SETQ S HOLD)
        (SETQ W NIL)
        (RETURN NIL))))
```

GENNAME is een functie die nieuwe namen schept. Een bekende standaardfunctie in LISP-systemen hiervoor is GENSYM, ook wel MKNAME (van make name). Deze vervullen de gevraagde functie van GENNAME.

Op de plaats waar bij de toestand Q3 in de functie ZNW ruimte opengelaten is kunnen nu de programma's voor de toestanden Q3, Q4 en Q5 ingevuld worden. Bij deze toestanden verandert dan (RETURN NIL) in (GO LOSE).

Boeiend is de term HOLD voor de zogenaamde 'hold list'. Dit is een extra geheugensteuntje voor RTN's en ATN's, waardoor dit type taalkundige analyse erg lijkt af te wijken van de wijze waarop mensen taal verwerken. Immers, met (SETQ HOLD S) wordt voor de aanvang van de ontleding eerst even de hele zin in het 'brein' opgeslagen, terwijl mensen taal begrijpen tijdens het aanhoren of lezen.

OPGAVEN

1. Ontwerp de functie UNION die de vereniging levert van de lijsten X en Y, dat wil zeggen dat de nieuw op te leveren lijst geen element dubbelop bevat indien dat element toevallig in X en Y beide mocht voorkomen. Dus voor (SETQ X '(AAP NOOT MIES)) en (SETQ Y '(NOOT MIES BOOM)) bevat (UNION X Y) de lijst (AAP NOOT MIES BOOM).
2. Definieer de functie DOORSNEDE die de doorsnede geeft van twee lijsten X en Y, dat wil zeggen die elementen oplevert die X en Y gemeenschappelijk hebben.
3. Gebruik de ideeën die gelanceerd werden bij het zinnen genereren in stille uren, namelijk het inlezen van complete grammatica's om te begrijpen wat Winston schrijft en programmeert in zijn boek 'Artificial Intelligence' over een compiler voor het construeren van augmented transition networks (= ATN's).

3.3.2.3 Een patroonherkennende parser

Zinsontleden wordt in de informatica parsen genoemd. De definitie van parsen is dan meestal in termen vervat als: een gegeven input string moet worden gematcht tegen een serie herschrijfsymbolen in een grammatica. Dit idee zit onder alle formele grammatica's en de implementaties ervan. Het idee is: geef alle zinsstructuren op die toegestaan zijn in een taal en laat de computer dan maar uitzoeken of een gegeven zin toegestaan is in de taal. Van dit idee wordt aangenomen dat het ook echt zo werkt bij mensen. Dat mensen dus wachten tot ze alles gehoord hebben. Dan nagaan of wat ze gehoord hebben in de grammatica past. Vervolgens constateren dat de zin grammaticaal is. Om tenslotte te gaan bedenken wat de zin dan wel zal betekenen.

Wanneer men het proces van taalverwerking in de praktijk waarneemt valt op, dat mensen niet wachten tot zinnen uit zijn en dat ze dan tóch dikwijls al weten wat er bedoeld wordt. Dit wijst erop dat het niet zo kan zijn dat een mens moet wachten tot hij alles tegen zijn grammatica heeft kunnen aanhouden. Het wijst erop dat het proces van taalverstaan niet bestaat uit een serie stappen die achter elkaar worden afgewerkt, maar veeleer uit een serie handelingen die zo snel mogelijk leiden naar het gewenste resultaat, dat wil zeggen begrijpen wat een ander zegt en/of schrijft. Om dit te kunnen doen moet het mogelijk zijn alle informatie die overbodig is weg te sturen. Anders gezegd: wanneer men zelf al kan invullen wat er gaat komen, hoeft men zijn aandacht niet meer op dat stukje te richten, maar men kan dan die aandacht gebruiken om op andere zaken te letten. Kunnen invullen wat er gaat komen, kan alleen wanneer men een grote ervaring heeft en/of kennis van wat er gaat komen. Anders gezegd: men weet de patronen van de zinnen, misschien zelfs van de gedachten en controleert slechts of aan de patronen wel wordt voldaan.

Dit nu is een pleidooi voor het ontwikkelen van een programma dat zinnen ontleedt op grond van de te verwachten patronen, niet op grond van een vaste voorraad vooraf in een trommeltje opgeslagen hele zinsstructuren. Daarbij komt dat men hiermee het back tracken kan voorkomen, hetgeen leidde tot de zogenaamde combinatorial explosion.

- METEOR voor zinsontleding

De patroonherkenner die we gebruiken is METEOR. De grammatica's die we tot nu toe definieerden waren in CF vorm. Bijvoorbeeld:

```
S    ::= NP+VP
NP   ::= DET+ADJ+N
VP   ::= VB+NP
DET  ::= 'de', 'het', 'een'
ADJ  ::= 'oude', 'jonge', 'glimmende'
N    ::= 'vis', 'man', vrouw'
VB   ::= 'ziet', 'vangt', 'eet'
```

Met deze BNF grammatica kunnen we de volgende zinnen ontleden:
- de oude man vangt de glimmende vrouw
- de jonge vis ziet de oude vrouw
- de oude vrouw eet de glimmende vis

De METEOR rules voor de ontleding van de eerste zin zijn de volgende:

1. Replacement voor woordenboek-consultaties:

```
(*     ( DE )          ( *K DET / 1 )     /)
(*     ( OUDE )        ( *K ADJ / 1 )     *)
(*     ( MAN )         ( *K N   / 1 )     *)
(*     ( VROUW )       ( *K N   / 1 )     *)
(*     ( GLIMMENDE )   ( *K ADJ / 1 )     *)
```

2. Compressie tot lijsten voor grammaticale analyse:

```
(*     ( DET $ N )     ( *K NP / 1 2 3 )     /)
(*     ( VB  NP  )     ( *K VP / 1 2   )     *)
(*     ( NP  VP  )     ( *K S  / 1 2   )     *)
```

3. Regels om zinsdelen te benoemen.

Wat tot nu toe buiten beschouwing bleef was het feit dat men normaal bij zinsontleding ook nog de zinsdelen, bijvoorbeeld onderwerp en gezegde, benoemt.

In de voorbeeldzinnen die we hebben gekozen is het onderwerp altijd de eerste NP. Het gezegde is de VB, het lijdend voorwerp is de tweede NP. Om nu onderwerp, gezegde en lijdend voorwerp in deze zinnen te vinden moeten we ingrijpen onmiddellijk nadat de NP's in de zinnen zijn gevonden, dat wil zeggen onmiddellijk na de eerste regel van het compressiedeel.

Voor het onderwerp ziet de regel er als volgt uit (we schrijven het onderwerp weg naar de Shelf OND):

```
(*     ( NP )     (/  (*Q OND 1 )     *)
```

Het lijdend voorwerp vinden we door de NP rechts van de VB te matchen, dat wil zeggen tegelijk met de VP match, als volgt:

```
(*   ( VB NP )   (*K VP / 1 2 )  (/  (*Q OBJ 2 )   *)
```

Ofwel in enkele regels is de hele zinsontleding voor dit type zinnetjes gedaan in METEOR.

OPGAVE. Schrijf de METEOR regels voor de andere voorbeeldzinnen.

- Snel een woordenboek raadplegen in METEOR

Woordenboek-consultatie kan gaan via directe match. Dat is nuttig en gemakkelijk wanneer het gaat om een klein bestand van woorden. Moet men een groot aantal woorden laten opzoeken dan zijn daarvoor speciaal ontwikkelde snelle technieken. We geven er hier een van. De programmatekst werd geschreven door D. Bobrow, de ontwerper en uitvinder van METEOR. Bij publiceerde die tekst in het boek dat hij samen met Berkeley uitgaf (zie literatuurlijst).

```
METEOR ((
        (* DICT ( JONGEN (( JONGEN / N HIJ )))
                ( MEISJE  (( MEISJE  / N ZIJ ))))

        ( ZOEKOP (( WOORD ($.1))) 0 (/  (*Q PAK
                   (FN GETDCT WOORD DICT))
                   (*P PAK )) ZOEKOP)
                   (* ( $ )     ((*A PAK ))
          END))
```

Als men nu een tekst geeft waarin de woorden jongen en meisje voorkomen, geeft deze procedure voor iedere jongen en voor ieder meisje de volgende definitie:

```
( JONGEN / N HIJ )      ( MEISJE / N ZIJ)
```

Op deze wijze heeft men van alle zelfstandige naamwoorden behalve de aanduiding dat het zelfstandige naamwoorden zijn ook nog de aanduiding dat ernaar moet worden verwezen met HIJ of met ZIJ. Zo kan men natuurlijk elke soort informatie meegeven in de woordenboekdefinities. Ofwel: men heeft op zeer elegante wijze een woordenboek geprogrammeerd.

3.3.2.4 Ontwerp van een Deep Case Grammatica

Taal begrijpen zonder de betekenis van de uiting en de woorden te weten is een onmogelijkheid. Van de andere kant is het in de taalkunde uiterst moeilijk gebleken om een theorie te bedenken waarin het mogelijk is de betekenis van uitingen ook op overzichtelijke en formele wijze weer te geven. Dat was de reden dat bijvoorbeeld Chomsky het eigenlijk liever liet afweten als het over betekenissen ging en zich concentreerde op een systeem waarin kon worden gecontroleerd of de structuren van de zinnen wel goed waren. Toch heeft men vanaf Plato al lang en uitgebreid nagedacht over hoe het nu zit met woorden en betekenissen. Het denken daarover heeft geleid tot een systeem dat de betekenis-eenheden in het Latijn beschreef. Dit systeem is in feite nooit overtroffen, het wordt aangeduid als 'de Latijnse grammatica'. Geen taalkundige theorie, zelfs de meest

geavanceerde, ontkomt aan de basisideeën van de Latijnse grammatica.

• Betekenisbeschrijving in de Latijnse grammatica

Een belangrijk aspect van de betekenis van de zin wordt gevormd door het werkwoord. Het werkwoord in het Latijn kan onder meer de volgende modaliteiten uitdrukken:
- de tijd waarin iets gebeurt (heden, verleden of toekomst)
- het aspect ten aanzien van het werkwoord: de voltooid of onvoltooid verleden tijd
- vorm: enkelvoudig, samengesteld
- soort: stellend, bevelend, vragend
- essentie: positief, negatief (met het woord 'niet' dus), onbepaald
- modus: mogen, willen, kunnen, moeten
- wijze: hiervoor gebruiken wij een zogenaamd bijwoord (g o e d gedaan, etc.)
- tijd: ook hiervoor gebruiken wij bijwoorden (kom m o r g e n)

Bij de rubrieken van 'wijze' en 'tijd' zijn we aangekomen bij implicaties ofwel verdere informatie bij een werkwoord.

OPGAVEN. Breng de volgende zinnen (waar mogelijk) onder bij de opgestelde categorieën.
1. Hij brak zijn been.
2. Ik zal komen.
3. Je mag je wel eens wat beter uitdrukken.
4. Door niet willen komt men niet verder.
5. Het regent hier nooit.
6. Wil je een stuk chocolade?
7. Overmorgen is het te laat.

• Implicaties van een werkwoord

Een werkwoord staat niet zomaar los in de ruimte. Men zegt niet: 'Brak' als men een volledige mededeling wil doen, maar men zegt bijvoorbeeld: "Kees brak zijn been."

of: "Kees brak de ruit met een stuk hout."

Men zou deze laatste zin als volgt kunnen omschrijven: Kees is een handelende persoon (Actant), het gaat om een ruit (Thema); de handeling die wordt verricht geschiedt door middel van een stuk hout. Er zijn om zo te zeggen eigenlijk twee handelende eenheden, namelijk Kees en het stuk hout. Men zou dus het werkwoord breken kunnen omschrijven als een handeling met een thema en twee actanten. Wanneer men zo tegen de zaak aankijkt, bevindt men zich in wat genoemd wordt de diepte casus grammatica (deep case grammar). Deze grammatica neemt de terminologie uit de oude Latijnse grammatica weer op en plakt er een nieuw etiket op (namelijk 'deep case grammar').

Deze deep case grammar werd in 1968 (!) 'uitgevonden' door Fillmore. Alles wat met handelingen en handelende personen etc. te maken heeft wordt in deze grammatica 'Case' genoemd. In dit geval Casualactant. Kees brak de ruit met een stuk hout kan men nu als volgt schrijven:

Breken: CA1 Kees, Thema ruit, CA2 een stuk hout
(CA1 = casualactant nummer 1)

Behalve thema kan men ook nog hebben plaats (locus).
Marie loopt in het park
Lopen: CA1, locus in het park

Men kan ook twee thema's of 2 loci hebben. Bijvoorbeeld:
Een aap is een dier
Zijn: T1 een aap, T2 een dier
(T1 = thema 1)
Marie draagt een regenmantel in de keuken
Dragen: CA1 Marie, T1 een regenmantel, T2 de keuken

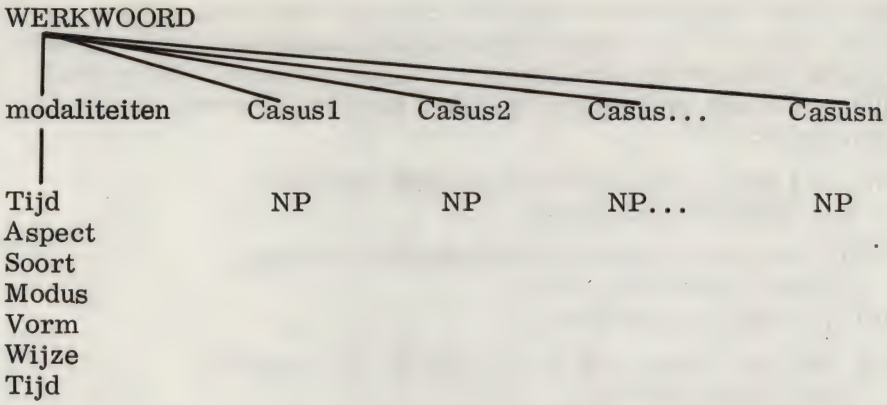
Behalve thema en actant heeft men nog andere categorieën, bijvoorbeeld doel, richting of instrument. Bijvoorbeeld:
Piet rent naar school
Rennen: CA1 Piet, doel naar school

De zin: Piet rent naar school heeft verder de volgende kenmerken:
Tijd: heden (ook wel genoemd: tegenwoordige tijd)
Wijze: actief; Vorm: enkelvoudig; Essentie: positief;
Soort: mededelend

OPGAVE. Benoem de volgende zinnen als diepte casus gevallen (zie: Marie draagt een regenmantel in de keuken):
1. Jan gaat naar bed
2. Jan gaat met Marie naar bed
3. Jan gaat met de lift naar bed
4. Marie zit in de tuin
5. Marie zit met een breiwerkje in de tuin

Met het plaatje op p. 176 kunnen we het geheel overzichtelijk maken.

Sleutelwoorden bij modaliteiten:
Tijd: heden, verleden, toekomst
Aspect: voltooid, onvoltooid
Soort: stellend, bevelend, vragend
Essentie: positief, negatief, onbepaald
Modus: mogen, willen, kunnen, moeten
Wijze: bijwoord
Tijd: bijwoord



- Programmeren van een DEEP CASE GRAMMATICA in LISP

Een programma voor de VP (het draait immers om de werkwoordsgroep, in het Engels afgekort VP) deep case grammatica ziet er als volgt uit:

```
(Q10(PUSH VP T

(VP(CAT AUX(GETF BE)
    (MAKEPR (QUOTE NBR)(GETF NBR))
    (ADDPR (QUOTE TENSE)(GETF TENSE))
    (ADDPR (QUOTE ASPECT)(QUOTE IMPERF))
    (ADDPR (QUOTE MOOD)(QUOTE INDIC))
    (TO V1))

  (CAT AUX(GETF(HAV)
    (MAKEPR (QUOTE NBR)(GETF NBR))
    (ADDPR (QUOTE TENSE)(GETF TENSE))
    (ADDPR (QUOTE ASPECT)(QUOTE PERF))
    (ADDPR (QUOTE MOOD)(QUOTE INDIC))
    (TO V4))

  (CAT V T
    (MAKEPR (QUOTE NBR)(GETF NBR))
    (ADDPR (QUOTE TENSE)(GETF TENSE))
    (ADDPR (QUOTE VOICE)(QUOTE ACTIVE))
    (ADDPR (QUOTE ASPECT)(QUOTE IMPERF))
    (ADDPR (QUOTE MOOD)(QUOTE INDIC))
    (JUMP V6)))

(V1(CAT V (GETF +ING)
    (ADDPR (QUOTE VOICE)(QUOTE ACTIVE))
    (ADDPR (QUOTE FORM)(QUOTE PROGRESSIVE))
    (JUMP V6))
```

```
  (CAT V (GETF +EN)
    (ADDPR (QUOTE VOICE)(QUOTE PASSIVE))
    (ADDPR (QUOTE FORM)(QUOTE SIMPLE))
    (JUMP V6) ))

(V4(CAT V(GETF +EN)
    (ADDPR (QUOTE VOICE)(QUOTE ACTIVE))
    (ADDPR (QUOTE FORM)(QUOTE SIMPLE))
    (JUMP V6))
    (CAT AUX T
    (TO V5)))

(V5(TST VP T
    (LIFTR ARGS (GETF ARGS))
    (LIFTR PDIGM (GETF PDIGM))
    (MAKEPR (QUOTE MODAL)(PUTPRL(GENSYMC)(GETR MLIST)))
    (ADDPR (QUOTE TOK) *)
    (TO V7) ))

(V7(POP (GETR MLIST)T)) ))
```

3.3.2.5 Bloemen voor Patricia Hearst of: Woorden benoemen per computer

In de voorafgaande paragrafen hebben we steeds een woordenboek gebruikt om woorden en de betekenissen ervan op te halen. De moeilijkheid daarbij was steeds dat een woord niet eenduidig is. Ook in het barre leven loopt men tegen deze eigenschap van woorden aan. Zo vindt men in de literatuur een reeks anekdotes over geplaagde kandidaten op MO-A examens Nederlands die niet overweg kunnen met de meerduidigheid van woorden.

Eén ervan komt uit het gedicht 'De ziele betracht de nabijheid Gods' van Jan Luycken. De regels 11 t/m 13 van dat gedicht luiden:

> Was nu alle ongelykheid voort
> En 't herte rein gelyk het hoord,
> geen hoogte, noch geen diepte zouw ons scheiden.

Veel van de kandidaten MO-A Nederlands die deze tekst kregen voorgelegd begrepen het eerste woord niet en verklaarden het gedicht als "nu moet je alle ongelijkheid wegwassen en het hart rein", in plaats van dat ze de verzuchting van de tekst begrepen: was er nu maar geen ongelijkheid en was het hart maar zuiver, dan zouden we niet gescheiden zijn. Als je zou zeggen dat er veel Nederlanders zijn die geen onderscheid kunnen maken tussen 'wassen' en 'zijn', zou je op zijn minst ongelovig worden aangekeken.

Een ander voorbeeld gaat om de beroemde tekst die Vondel schreef voor zijn vriend Geeraert Vossius. Daarin komt de zin voor:

> Beny uw soon de hemel niet,
> De hemel treckt; ay, laat hem los.

Er was een kandidaat die het woord 'ay' opvatte als een eigennaam en dus dacht dat er stond "Arie, laat hem los" (Ay is een in het Nederlands voorkomende eigennaam voor Arie). Het ging in deze gevallen om kandidaten die $2\frac{1}{2}$ uur de tijd kregen en al meer dan twintig jaar via hun moeder, het onderwijs, de radio, de TV, enz. waren blootgesteld aan de Nederlandse taal.

Nu is er geen woord te bedenken dat niet minstens twee verschillende betekenissen heeft; een enkele blik in Koenen of Van Dale leert zelfs dat er woorden zijn met meer dan vier betekenissen. Een woord als 'helder' of 'hemel' bijvoorbeeld.

Wil men de computer inzetten bij analyses van taal, dan zal men een oplossing voor het probleem van de meerduidigheid van woorden moeten vinden. Nu hebben we het nog niet eens gehad over meerduidigheid die verborgen is in de ironie wanneer men bijvoorbeeld zegt: "Mooi weertje hebben we vandaag", terwijl men bedoelt: "het is pokkeweer". Voor het oplossen van dit soort hogeschool-meerduidigheid waartoe mensen met taal in staat zijn, is naar ons weten nog geen oplossing per computer gevonden. Hoe het komt dat juist deze vormen van meerduidigheid, die zo belangrijk zijn voor echte communicatie, niet aan de orde zijn gekomen, kan worden verklaard uit de geschiedenis van de 'gewone' woorden.

- De optel- en vermenigvuldigstrategie

De computer komt oorspronkelijk van rekenaars. Wiskundigen waren dan ook de eersten die dit apparaat in hun vak gingen gebruiken. Daarna volgden de andere 'exacte' vakken. Pas in de vijftiger jaren begon men ook schoorvoetend te kijken naar computers en taal. Men had toen een naïeve opvatting over taal. Taal werd beschouwd als een verzameling woorden waaruit een verzameling zinnen kon worden gemaakt. Daarom werden pogingen in het werk gesteld om het aantal betekenissen van woorden in getallen te vangen om zodoende een overzicht te krijgen van hetgeen men in de computerprogramma's moest inbouwen. Dit leidde tot merkwaardige conclusies. Zo kwam een exact onderzoeker van de Duitse taal tot de conclusie dat het Duitse lidwoord 57 betekenissen had. Dat kwam doordat hij alle naamvallen, zowel van mannelijke, vrouwelijke als onzijdige woorden meetelde. Ook voor het Nederlands zijn soortgelijke beweringen gedaan, trouwens voor alle talen.

Nu hebben taalkundigen nogal eens de neiging de meest rare zinnen te verzinnen ter adstructie van wat ze willen beweren. Zo ook de exact ingestelde computer-taalkundigen. Een beroemde zin uit deze periode luidt (we vermeldden hem reeds):

> Als vliegen achter vliegen vliegen dan vliegen vliegen achter vliegen.

(Vliegen spelen trouwens toch een belangrijke rol in taalkundige discussies.)

Van de vliegende-vliegen-zin werd beweerd dat het woord vliegen de volgende betekenissen had: zelfstandig naamwoord eerste t/m derde persoon meervoud; werkwoord aantonende wijs eerste t/m derde persoon meervoud en onbepaalde wijs. Dat wil zeggen: 4+3+1 = 8. Het woord vliegen komt in deze zin zes maal voor, dus alleen al op dit niveau van analyse heeft de zin 6x8 = 48 betekenissen. Daarbij is dan niet uitgezocht dat het woord vliegen zelf ook nog zeven betekenissen heeft als werkwoord volgens Koenen en dat het als zelfstandig naamwoord behalve naar een insekt ook nog in allerlei uitdrukkingen naar andere betekenissen verwijst. De woordjes 'als' en 'dan' uit de zin hebben ook nog een paar betekenissen. Als men al die betekenissen met elkaar vermenigvuldigt komt men al snel uit op enige honderden betekenissen per zin. De conclusie die men uit de observaties bij deze zin trok was, dat de meerduidigheid van taal zulke astronomische proporties aanneemt dat ze voor onoplosbaar moet worden gehouden. Daarmee werd het probleem als afgedaan beschouwd en een zich respecterend taalkundige houdt de redenering nog steeds staande. Men is algemeen van mening dat automatisch taalkundige coderingen aanbrengen in zinnen en teksten onmogelijk is vanwege de complexiteit van de taal. De meeste taalkundigen hebben zich daarbij nooit afgevraagd hoe het dan komt dat een normale gebruiker geen last heeft van de astronomische druk van meerduidigheid. Het zou toch op zijn minst een wonder moeten worden genoemd dat een normale Nederlander die vliegende-vliegen-zin direct begrijpt. Het is zelfs zo dat geen Nederlander behalve een taalkundige ook maar op de gedachte komt dat die zin meer dan één, laat staan een paar honderd betekenissen zou hebben.

Doordat men niet verwonderd was over dit feit dat een taalgebruiker direct een betekenis weet kwam men ook niet tot de ontdekking dat de redenering fout was. De basis ervan is fout. Taal bestaat niet uit een verzameling woorden of zinnen die ieder een los bestaan leiden wat hun betekenissen betreft. Taal begrijpen is niet het opvoeren van tientallen mogelijke of onmogelijke betekenissen waarna men dan wel gaat kiezen wat het meest past. Het is veeleer zo dat een taalgebruiker direct een betekenis weet, kwam men ook niet tot de ontdekking dat de redenering fout was. De basis ervan is fout. Taal bestaat niet uit een verzameling woorden of zinnen die ieder een los bestaan leiden maken over wat er gaat komen. Taal verwerken moet voor een gedeelte geschieden aan de hand van verwachtingspatronen. Tot zover de hypothese of de theorie.

In onderzoek op het gebied van de taalpsychologie bleek dat luisteraars en lezers inderdaad gebruik maken van de verwachtingen die ze op grond van de kennis van hun taal hebben. De conclusie lag dus voor de hand dat een computerprogramma dat meerduidigheid van woorden moet oplossen, ook gebruik moest maken van die 'taalkundige' ver-

wachting van de lezer of luisteraar.

Een tweede inzicht dat de analyse van het probleem opleverde betrof de meerduidigheid zelf. Er zijn namelijk twee soorten meerduidigheid. De eerste soort is verbonden met de woordklasse waartoe een woord behoort. Het verschil in betekenis tussen 'was' in: "was je gisteren thuis" en 'was' in: "was je je auto vandaag?" kan worden opgelost, wanneer de computer kan vaststellen dat het eerste 'was' een koppelwerkwoord is en het tweede 'was' een zelfstandig werkwoord. In de zin "Heb je de was aan huis?" vindt de computer dat in deze zin 'was' een zelfstandig naamwoord is. De eerste klasse van meerduidigheid is dus de meerduidigheid die wordt veroorzaakt doordat woorden (liever woordtekens of woordvormen) tot meer dan een woordklas kunnen behoren (zogenaamde homografen). Een computerprogramma dat de juiste woordklassen vindt, filtert deze meerduidigheid uit.

De tweede soort meerduidigheid betreft de betekenissen van een woord binnen één en dezelfde woordklasse (zogenaamde homoniemen). Als zelfstandig naamwoord heeft 'was' bijvoorbeeld nog de betekenissen die uit de volgende zinnetjes blijken:

Er is was op de bovenrivier
Ik heb de was aan de lijn
Ik gebruik zelden bijenwas voor kaarsen

De verschillende betekenissen van het zelfstandig naamwoord 'was' kunnen alleen worden afgeleid uit de betekenissen van de woorden die eromheen staan. De rivier in de eerste zin bepaalt wel dat 'was' hier niet zal betekenen 'vuil goed', terwijl de 'lijn' in de tweede zin nu net wel ervoor zorgt dat men denkt aan vuil goed. De kaarsen in de derde zin zorgen ervoor dat men niet denkt dat de bijen de was doen.

- Een patroonherkennende parser voor woordklastoekenning

De kunst is dus een computerprogramma zo te maken dat er direct kan worden gekozen en er niet een structuurboom moet worden gemaakt waarin men dan heen en weer moet fietsen wanneer het niet goed gaat (back tracking en de combinatorial explosion!). Anders gezegd: maak een computerprogramma waarin de schijnmeerduidigheden niet worden gemaakt. Zoals iedere keer wanneer we bij dit soort problemen aankwamen, is het ook hier weer de moeite waard een patroonherkennend programma te schrijven waarin de verwachtingen van de taalgebruiker zitten.

- Verwachtingen ten aanzien van woordklassen

Er zijn sinds de Latijnse grammatica volop tot bloei was gekomen de volgende woordklassen onderkend:

lidwoorden	(de, het, een)
bijvoeglijke naamwoorden	(mooie, oude)
bijwoorden	(dat staat je goed)
zelfstandige naamwoorden	(huisje, boompje, beestje)
eigennamen	(Jantje, Pietje, Keesje)
werkwoorden	(staan, lopen, zitten, schrijven)
hulpwerkwoorden	(zijn, moeten)
persoonlijke voornaamwoorden	(ik, jij, hij)
bezittelijke voornaamwoorden	(mijn, dijn, zijn)
betrekkelijke voornaamwoorden	(de man die daar loopt)
aanwijzende voornaamwoorden	(die man is gek)
vragende voornaamwoorden	(wie, wat)
onbepaalde voornaamwoorden	(alle, vele, sommige)
voorzetsels	(in, op, bij)
voegwoorden	(toen, nadat, omdat)

Deze woordklassen kan men onderscheiden in twee klassen die een volstrekt verschillende functie hebben. Zie bijvoorbeeld de volgende zin:

Patricia kreeg bloemen, toen ze het proces gewonnen had.

In deze zin staat een serie mededelingen te lezen over 'de werkelijkheid', over wat er gebeurde en met wie. Deze mededelingen vindt men in de woorden

-Patricia
-kreeg
-bloemen
-proces
-gewonnen

De andere woorden als 'ze' en 'het' dienen alleen maar om de structuur van de zin aan te geven, of om verwijzingen binnen de tekst te geven. Het woordje 'toen' heeft een dubbele functie. Ten eerste geeft het een tijdsaanduiding aan en ten tweede geeft het aan dat er nu een bijzin komt. Om deze dubbele functie met zekerheid te onderkennen moeten we dit 'toen' echter wel kunnen onderscheiden van het bijwoord 'toen':

Toen zakte hij door zijn stoel.

VRAAG. Om welk toen gaat het:
Toen hij dat hoorde, toen werd hij pas goed kwaad.

Samenvattend kan men dus stellen dat er twee totaal verschillende soorten woorden in een tekst aanwezig zijn: de zogenaamde structuurwoorden, ook wel grammaticale woorden genoemd, en de zogenaamde

inhoudswoorden. De structuurwoorden geven de structuur van de zin aan. Voor onze patroonherkenner kan men dat omkeren en zeggen: als er een structuurwoord aankomt, heb ik ook de rest van de structuur te verwachten. Concreet: als je tegen een Nederlander zegt: 'de' dan wacht hij tot er een zelfstandig naamwoord komt. Dit nu kunnen we gebruiken voor de patroonherkenner die woordklassen moet vinden.

Van deze wetenschap uit nu kan men een patroonherkenner definiëren die de woordsoorten benoemt.

- Hoe de vliegen vliegen in METEOR

Het programma in METEOR dat de vliegende vliegen onderkent als werkwoord en zelfstandig naamwoord heeft de volgende onderdelen.

1. Een woordenboek-onderdeeltje waarin de structuurwoorden zijn opgenomen.
2. Een patroonherkennend gedeelte waarin aan de hand van de structuurwoorden en de reeds erbij gevonden inhoudswoorden de ontbrekende inhoudswoorden worden aangevuld.

1. Woordenboekgedeelte

De zin was: Als vliegen achter vliegen vliegen, dan vliegen vliegen achter vliegen.

In deze zin komen de volgende structuurwoorden voor:

als: voegwoord, met een van het Latijn afgeleid woord 'conjunctie' (hetgeen we afkorten tot CON)

achter: voorzetsel, met een van het Latijn afgeleid woord 'prepositie' (hetgeen we afkorten met PRP)

In METEOR wordt deze informatie via eenvoudige matching aangebracht:

```
(*     ( ALS )      (  (*K  CON / 1 )     *)
(*     ( ACHTER )   (  (*K  PRP / 1 )     *)
```

2. Patronen

Tussen 'als' en 'achter' staat één woord. Dat kan slechts een zelfstandig naamwoord zijn.
De METEOR regel:

```
(*     ( CON  ($.1)  PRP)     ( 1 (*K / SUBST ) 2 )     *)
```

Het voegwoord 'als' verlangt een werkwoord en wel daar waar de bijzin uit is. In ons geval is dat te zien aan de komma.
De METEOR regel:

```
(*  ( CON  $  ($.ATOM)  (* COMMA ) )  ( 1 2 (*K VB / 3) 4)  *)
```

De toestand is nu:

(CON / ALS) (SUBST / VLIEGEN) (PRP / ACHTER)
VLIEGEN (VB / VLIEGEN) , DAN VLIEGEN VLIEGEN
(PRP / ACHTER) VLIEGEN.

Tussen de prepositie 'achter' en het verbogen werkwoord (VB) staat slechts één woord. Dit woord kan slechts een zelfstandig naamwoord zijn, want een voorzetsel vereist een zelfstandig naamwoord.
De METEOR regel:

(* (PRP ($.ATOM) VB) (1 (*K SUBST / 2) 3) *)

De toestand is nu als volgt:

(CON / ALS) (SUBST / VLIEGEN) (PRP / ACHTER)
(SUBST / VLIEGEN) (VB / VLIEGEN) , DAN VLIEGEN VLIEGEN
(PRP / ACHTER) VLIEGEN.

De plaats achter de komma is niet direct zeker, omdat daar zowel een werkwoord kan staan als een bijwoord zoals hier. We moeten dus eerst rechts een houvast zoeken. Dat vinden we in de eerste prepositie. Het woord dat daarvoor staat moet een zelfstandig naamwoord zijn. Links daarvan staat het werkwoord, links daarvan het bijwoord.
De METEOR regel is aldus:

(* (CON $ (* COMMA) ($.ATOM) ($.ATOM) ($.ATOM) PRP)
(1 2 (*K ADV / 3) (*K VB / 4) (*K SUBST / 5) *)

De toestand is aldus:

(CON / ALS) (SUBST / VLIEGEN) (PRP / ACHTER)
(SUBST / VLIEGEN) (VB /VLIEGEN) , (ADV / DAN)
(VB / VLIEGEN) (SUBST / VLIEGEN) (PRP / ACHTER) VLIEGEN.

Er is nu slechts één regel nodig. Namelijk om te kunnen toekennen voor het geval er een voorzetsel staat met daarachter nog maar een enkel woord. Dit woord moet een zelfstandig naamwoord zijn.
De regel luidt aldus:

(* PRP ($.ATOM) ($.0)) (1 (*K SUBST / 2)) *)

Het hele programma voor dit soort zinnen ziet er in METEOR dus als volgt uit:

```
(*  (ALS)       (  (*K CON / 1)  *)
(*  (ACHTER)  (  (*K PRP / 1)  *)
(*  (CON $ ($.ATOM) (* COMMA) ) (1 2 (*K VB / 3) 4)  *)
(*  (CON ($.ATOM) PRP) (1 (*K SUBST / 2) 3)  *)
(* (PRP ($.ATOM) VB) (1 *K SUBST / 2) 3)  *)
(*  (CON $ (* COMMA) ($.ATOM) ($.ATOM) ($.ATOM) PRP)
    (1 2 (*K ADV / 3) (*K VB / 4) (*K SUBST / 5)  *)
(*  (PRP ($.ATOM) ($.0) ) (1 (*K SUBST / 2)  *)
```

Ofwel: de als onoplosbaar gekenmerkte zin is opgelost in zeven regels.

- Toepassingen

In eerste instantie zou men kunnen denken dat het toekennen van woordklassen en het vinden van woordbetekenissen een soort intelligent spelletje is voor taalkundigen en dat de resultaten alleen maar van belang zijn voor hen die puur wetenschappelijk zijn ingesteld. Niets is echter minder waar.

Alle vormen van kwantitatief taalonderzoek met de computer zijn in het verleden vastgelopen op het feit dat de computer woorden bij elkaar optelt die een verschillende betekenis hebben, maar op dezelfde wijze geschreven worden. Dit onderzoek is van belang voor het onderwijs, bij intelligentieonderzoek, bij het onderzoek naar taalachterstand. Maar ook de automatische vertaalmachine, de automatische inhoudsanalyse van allerlei teksten kan niet zonder een computerprogramma dat de meerduidigheid van woorden oplost. Verder wordt dit soort programma's gebruikt in de rechtspraak. Het schijnt namelijk zo te zijn dat men anonieme briefschrijvers kan achterhalen door de manier waarop woordklassen naast elkaar worden gebruikt. Dat schijnt namelijk per taalgebruiker verschillend te zijn. In ieder geval zijn computerberekeningen op woordklascombinaties in het proces van Patricia Hearst gebruikt om aan te tonen dat ze bepaalde brieven niet zelf had geschreven. Verder is er een theorie dat afasie, een beschadiging van het taalcentrum in de hersenen, kan worden beschreven aan de hand van de woordklassen. En last but not least: een computer die woordklassen kan vinden, kan ook in het onderwijs worden ingezet om aan anderen te leren hoe dat moet. Als men beseft dat veel leerkrachten vinden dat het grammatica-onderwijs een bescheidener plaats moet krijgen terwijl ze dit onderwijs toch niet willen afschaffen, kan men rustig de conclusie trekken dat hier eveneens een mooie toepassing ligt voor de computer die woorden kan benoemen.

3.3.3 Inhoudsanalyse

In de loop van dit boek is het betoog langzaam verschoven van wat werd genoemd de fysieke structuur van de tekst naar wat werd genoemd de inhoud. Daarbij kwamen aan de orde het maken van woordenlijsten tot het maken van grammatica's en het opstellen van een intelligente data base. Ook zaken als woordafbreken en geschreven woorden omzetten in gesproken woorden kwamen aan de orde. We bleven daarbij echter steeds binnen het kader van de zin of van het woord.

Wil men zich met de inhoud van een tekst bezig houden dan bestaat taal niet meer uit woorden of zinnen. Of liever: misschien bestaat taal dan juist wel uit woorden en niet meer uit zinnen. Althans men zal het nodige moeten doen om van de ene zin naar de andere te komen, dat wil zeggen om het verband daartussen te zien. Nemen we de volgende dialoog die werd verricht door het samenspel van mens en computer (m = mens, c = computer). De dialoog gaat tussen een computer die handelingen kan verrichten; hij kan blokken opstapelen en verplaatsen. Ook kan hij vragen beantwoorden. Hij is dus geprogrammeerd als robot. De dialoog die hier wordt weergegeven heeft echt plaatsgevonden. De voertaal was Engels, we geven een vertaling.

m: Pak een rood blok.
c: OK /d.w.z. hij begrijpt het en doet het/
m: Grijp de piramide.
c: IK WEET NIET WELKE PIRAMIDE JE BEDOELT.
/er waren namelijk drie piramiden aanwezig/
m: Zoek een blok dat groter is dan dat je nu vasthebt en zet hem in de doos.
/nu moet worden uitgezocht wat betekent: "dat je nu vasthebt"; de moeilijkheid is namelijk het verwijzende woord dat. Ook moet het verwijzende woord 'hem' worden opgelost/
c: IK NEEM AAN DAT JE MET "HEM" BEDOELT HET BLOK DAT GROTER IS DAN DAT IK NU VAST HEB.
m: Ja.
c: OK
m: Wat zit er nu in de doos?
c: DE BLAUWE PIRAMIDE EN HET BLAUWE BLOK
/de machine kan dus de geschiedenis van het vraag-antwoord-gebeuren onthouden; ook dat is nodig wanneer men inhoudsanalyse wil plegen/
m: Hoeveel blokken zitten er niet in de doos?
/moeilijkheid hier is het ontkennende woord 'niet', namelijk op welk gedeelte van de uitspraak slaat dat woord/
c: VIER
m: Is er tenminste eentje smaller dan dat je moest oppakken?
/deze vraag slaat helemaal terug op het eerste stuk, namelijk de opdracht: Pak het rode blok/

c: JA, DE RODE KUBUS
m: Staat die ergens op?
c: JA, OP DE TAFEL
m: Kan de tafel blokken optillen?
c: NEE
/dat wil zeggen de computer moet kennis hebben over de wereld waarover hij moet praten/
m: Kan een piramide op een andere geplaatst worden?
c: DAT WEET IK NIET
m: Kan een piramide op een blok staan?
c: JA
m: Zet de twee piramiden op elkaar.
c: DAT KAN IK NIET
/hij probeert het en het lukt niet!!/
m: De blauwe piramide is van mij.
c: IK BEGRIJP HET
/dat wil zeggen de computer slaat nieuwe informatie op/
m: Alle blokken die niet rood zijn, zijn van mij, maar iets is niet van mij als er een piramide op staat.
c: IK BEGRIJP HET
m: Is de doos van mij?
c: NEE
m: Zit er iets in de doos dat van mij is?
c: JA, TWEE DINGEN: HET BLAUWE BLOK EN DE BLAUWE PIRAMIDE
m: Wil je svp zowel de rode blokken als een groene kubus of een piramide op elkaar stapelen?
/het probleem hier is het 'zowel...als... of(wel)... of... en' probleem
c: OK
m: Welke kubus staat op de tafel?
c: DE GROTE GROENE WAAR DE RODE PIRAMIDE OP STAAT
m: Staat er een groot blok achter de piramide?
c: JA, DRIE: EEN GROOT ROOD, EEN GROTE GROENE KUBUS EN HET BLAUWE
m: Zet een kleine op de groene kubus die op de piramide staat
c: OK
m: Zet de kleinste piramide daar bovenop.
/het woord kleinste wordt afgeleid van de stam 'klein', kleinste zelf zit niet in het woordenboek van de computer/
c: OK
m: Hoeveel dingen staan er bovenop de groene kubussen?
c: HET IS ME NIET DUIDELIJK WAT JE BEDOELT MET: "BOVENOP" IN DE ZIN BOVENOP DE GROENE KUBUSSEN. BEDOEL JE:
- DIRECT OP DE OPPERVLAKTE
- HET DOET ER NIET TOE WAAR ER BOVENOP

/een van de grote problemen bij inhoudsanalyse is de betekenis van voorzetsels; denk aan zinnen als:
Hij liep met Mien door het park
Hij liep met een vaart door het park/

m: Heb je voor je de groene op de kleine kubus zette, een piramide aangeraakt?
c: JA, DE GROENE
m: Wanneer heb je die gepakt?
c: TOEN IK DE RODE KUBUS, EEN GROOT ROOD BLOK EN EEN GROTE GROENE KUBUS AAN HET OPSTAPELEN WAS
m: Waarom?
/moeilijkheid is hier: waar slaat dat waarom op?/
c: OM HEM KWIJT TE RAKEN
m: Waarom deed je dat?
c: OM DE RODE KUBUS VRIJ TE KRIJGEN
m: Waarom maakte je die rode kubus vrij?
/weer een verwijzend woord 'die': je moet maar weten dat dat slaat op 'de rode kubus' uit de vorige zin!/
c: OM HEM OP DE GROTE GROENE KUBUS TE ZETTEN
m: En waarom deed je dat?
c: OM DE RODE KUBUS, EEN ROOD BLOK EN EEN GROTE GROENE KUBUS OP ELKAAR TE STAPELEN
m: En waarom deed je dat?
c: OMDAT JE ME DAT VROEG
m: Noem het grootste blok superblok.
c: OK
m: Heb je sinds we begonnen iets gedaan met het superblok?
c: JA
m: Waarom liet je hem los?
c: IK NEEM AAN DAT JE MET "HEM" HET SUPERBLOK BEDOELT. IK KAN GEEN GEBEURTENISSEN UITLEGGEN DIE NIET HEBBEN PLAATSGEVONDEN
m: Dank je.
c: TOT JE DIENST

VRAGEN ter verwerking van de zojuist aangeboden kennis

1. Welke 'kennis van de wereld' heeft de computer nodig om deze dialoog te kunnen voeren?

2. Wat moet de computer tot zijn beschikking hebben om taal te begrijpen: a. op zinsniveau
   b. op meer dan zinsniveau

3. Welke categorieën van vragen komen in de dialoog voor?

4. Wat moet de computer tot zijn beschikking hebben om vragen te te beantwoorden?

5. Is het voldoende voor een goede dialoog wanneer de computer all alleen vragen kan beantwoorden?
6. Zou het ook mogelijk zijn dat de computer vragen in het Engels met Nederlandse zinnen beantwoordt?
   Wat zou er dan precies moeten veranderen?

• Een paar stukjes SHRDLU

Uit de specificaties boven is duidelijk geworden dat het niet eenvoudig zal zijn een echt programma te maken dat inhouden van teksten analyseert. De programmatekst van het programma waar een voorbeelddialoog van werd genomen omvat meer dan 1000 bladzijden. Het programma werd ontwikkeld door Terry Winograd (alias Tony Earwig) en is een van de broemdste computerprogramma's waarmee men natuurlijke taal kan verstaan. Om een indruk te geven hoe dat programma eruit ziet zijn in het onderstaande enige pagina's tekst van het programma van Winograd afgedrukt.

1. Woordenboekdefinities in SHRDLU

In het onderstaande worden definities gegeven van de volgende woorden:
"BALL" "BE" "BEFORE" "BEGIN" "BEGAN" "BEHIND" "BELOW" "BENEATH" "BESIDE" "BIG" "BLACK" "BLOCK" "BLUE" "BOTH" "BOX" "BRICK" "BUILD" "BUT" "BY" "CALL" "CAN" "CHOOSE" "CLEAN"

```
(DEFS BALL
   SEMANTICS ((NOUN (OBJECT
                      (MARKERS: (#MANIP #ROUND)
                       PROCEDURE: ((IS ***#BALL))))))
   FEATURES (NOUN NS))
(DEFS BE


   FEATURES (AUX VA AF INF)
   SEMANTICS ((VB
               ((THERE
                 (RELATION
                     (RESTRICTIONS: (((#THING)
                                      (EQ (QUANTIFIER? SMSUB)
                                          'INDEF)))
                      PROCEDURE: NIL)))
                (INT
                 (COND
                  ((RELATION
                       (RESTRICTIONS: (((#PHYSOB))
                                       (SMCOMP (#PROPERTY)))
```

```
                        PROCEDURE: (#EVAL
                                   (PROG (PROPERTY)
                                    (COND
                                     ((SETQ
                                       PROPERTY
                                       (MEFT (GET '#PROPERTY
                                                  ' SYSTEM
                                             (MARKERS? SMCOMP)))
                                      (RETURN
                                       (LIST (LIST (CAR PROPERTY)
                                                   '#1
                                                   '#2))))
                                     ((RETURN (LIST '(#2 #1)))))))))
                       (RESTRICTIONS: (((#THING))
                                       (SMCOMP
                                        (#SYSTEMS)
                                        (AND (NOT (REFER? SMCOMP))
                                             (EQ (REL? SMCOMP)
                                                 SMSUB))))
                        PROCEDURE: (#EVAL (RELATIONS? SMCOMP)))
                       (RESTRICTIONS: (((#THING))
                                       (SMCOMP (#THING)
                                               (REFER? SMCOMP)))
                        PROCEDURE: ((#EVAL
                                     (LIST 'THAMONG
                                           '#1
                                           (LIST 'QUOTE
                                                 (REFER? #2)))))))
                 (T (FRISTOP SORRY I DON'T UNDERSTAND THE
                             VERB BE WHEN YOU USE IT LIKE
                             THAT)))))))
(DEFS BEFORE SEMANTICS ((BINDER (SMBINDER NI: START))) FEATURES (BINDER TIME))
(DEFS BEGIN
    DEMANTICS ((VB
                '(TRANS (RELATION
                         (RESTRICTIONS:
(DEFS BEGAN IRREGULAR (BEGIN (PAST) (INF)))
(DEFS BEHIND
    SEMANTICS ((PREP (#IOC  AF    ))))
    FEATURES (PREP PLACE))
(DEFS BELOW
    SEMANTICS ((PREP (#IOC #ABOVE NIL)))
    FEATURES (PREP PLACE))
(DEFS BENEATH
    SEMANTICS ((PREP (#IOC #ABOVE NIL)))
    FEATURES (PREP PLACE))
(DEFS BESIDE
    SEMANTICS ((PREP (RELATION
                      (RESTRICTIONS: (((#PHYSOB)) ((#PHYSOB)))
                       PROCEDURE: ((#NEXTO #1  #2 #TIME))))))
    FEATURES (PREP PLACE))
(DEFS BIG
    SEMANTICS ((MEASURE (MEASURE DIMENSION:
                                 #SIZE
                                 RESTRICTIONS:
                                 (#PHYSOB)
                                 DIRECTION:
                                 T))
```

```
            (ADJ (OBJECT
                      (MARKERS: (#PHYSOB #BIG)
    PROCEDURE: ((#MORE  #SIZE***(128. 128. 128.))))))
    FEATURES (ADJ))
(DEFS BLACK SEMANTICS ((ADJ (#COLOR  #BLACK))) FEATURES (ADJ))
(DEFS BLOCK
    SEMANTICS ((NOUN (OBJECT
                      (MARKERS: (#MANIP  #RECTANGULAR)
                       PROCEDURE: ((#IS ***#BLOCK))))))
    FEATURES (NOUN NS))
(DEFS BLUE SEMANTICS ((ADJ) (#COLOR  #BLUE))) FEATURES (ADJ))
(DEFS BOTH
    SEMANTICS ((DET 'DEF))
    FEATURES (B-SPECIAL QNTFR DET DEF NPL BOTH)
    B-SPECIAL (BOTH AND)
    FEXPR (LAMBA (A)
          (PROG (END)
                (SETQ FND CUT)
                (RETURN (PROG (CUT NAB BOTH)
                              (SETQ NBR N)
                              (AND ( FLUSHMF)
                                   ( **N
                                      NR
                                      (FQ (WORD PTW) (CAR A))
                                      NW)
                                   (CUT END)
                                   (SETQ BOTH PTW)
                                   (SETQ RE
                                         (COND ((MEMQ (CAR REST)
                                                      '(PREP ADV))
                                                (PARSE 3 RES T))
                                               ((MEMQ (CAR REST)
                                                      '(NG PREPG
                                                        ADJG
                                                        CLAUSE))
                                                (PARSE 2 REST T))))
                                   (LESSP (LENGTH N) (LENGTH BOTH))
                                   (RETURN (SETQ SPECIAL 'SKIP)))
                              (SETQ BE NIL)
                              (SETQ N NBB)
                              (RETURN NIL))))))
(DEFS BOX
    SEMANTICS ((NOUN (OBJECT
                      (MARKERS: (#BOX)
                       PROCEDURE: ((#IS***#BOX))))))
    FEATURES (NOUN NS))
(DEFS BRICK FEATURES (NOUN NS))
(DEFS BUILD
    SEMANTICS ((VB ((TRANS (#BUILD)))))
    FEATURES (VB INF TRANS))
(DEFS BUT FEATURES (SPECIAL) SEMANTICS T SPECIAL (CONJ))
(DEFS BY
    SEMANTICS ((PREP (RELATION
                      (RESTRICTIONS: (((#PHYSOB)) ((#PHYSOB)))
                       PROCEDURE: ((#NEXTO #1 #2  #TIME))))))
    FEATURES (PREP))
```

```
(DEFS CALL
    SEMANTICS ((VB
                ((TRANS 2
                  (RELATION
                      (RESTRICTIONS: (((#ANIMATE)) ((#THING))
                                                   ((#NAME)))
                       PROCEDURE: ((#CALL #2  #3  #TIME))))))))
    FEATURES (VB INF TRANS 2))
(DEFS CAN SEMANTICS ((VR T)) FEATURES (V3P8 VFS VPL VB MODAL AUX))
(DEFS CHOOSE
    SEMANTICS ((VB ((TRANS (#NOTICE)))))
    FEATURES (VB INF TRANS))
(DEFS CLEAN SEMANTICS ((VB T)) FEATURES (VB INF VPRT TRANS))
```

2. In het onderstaande wordt het gedeelte van het programma SHRDLU afgebeeld dat vragen beantwoordt.

```
(DEFUN ANSQUEST (RSS)
       ;;ANSQUEST ANSWERS ALL TYPES OF QUESTIONS BY SENDING THEM OUT
       ;;TO ANSREL OR ANSNOREL DEPENDING ON WHETHER THERE IS A REL.
       (COND
        ((OR (OR? RSS) (AND? RSS)) (PROG (ANS)
          (SETQ ANS (MAPCAR 'ANSQUEST
                            (OR (AND? RSS) (OR? RSS))))
          (RETURN
           (ANSBUILD
            (APPLY 'PLUS (MAPCAR 'PLAUSIBILITY? ANS))
            (APPEND
             (AND (NOT (ISQ (PARSENODE? RSS) COMPONENT))
                  '((SAY YOU/'RE TRYING TO CONFUSE ME/.)))
             (MAPCAN '(LAMBDA (QUEST) (APPEND
                '((TERPRI)) (ANSAY
                             (ELIZA
                              (FROM (NB (PARSENODE? (ANSRSS? QUEST)))
                                    (N (PARSENODE? (ANSRSS? QUEST))))))
       ;;CONJOINED QUESTIONS ARE HANDLED
       ;;REPEATING EACH PART AND ANSWERING
       ;;SEPARATELY
                        '((PRINC '?) (TERPRI))
                        (ACTION? QUEST)))
                  ANS))
                 NIL))))
        ((REL? RSS) (ANSREL RSS))
        (T (ANSNOREL RSS))))
```

Natuurlijk kan het programma slechts vragen beantwoorden over de blokkenwereld, maar toch kan het dat op heel competente wijze. Laten we eerst bestuderen waarover het in die blokkenwereld precies gaat:

- voorwerpen, zoals bepaalde blokken of piramiden
- betrekkingen tussen voorwerpen
- betrekkingen tussen voorwerpen en de eigenschappen ervan, zoals kleur en grootte
- handelingen, zoals het oppakken, opstapelen, enz.

De vragenbeantwoorder (= ANSwer QUESTions) beantwoordt alle soorten vragen (= RSS) door deze te geven òf aan de functie ANSREL indien er sprake is van een betrekking (= RELation) òf aan de functie ANSNOREL indien dat niet het geval is (= NO RELation). Om een betrekking te vinden in de vraag RSS kijkt het programma ANSQUEST zonodig herhaald met behulp van MAPCAR naar AND en OR functies in de formulering; zijn die er niet dan wordt de laatste regel geactiveerd, namelijk (T (ANSNOREL RSS)). Voordat een antwoord geformuleerd is op een vraag waarin een betrekking zit, wordt de nodige voorzichtigheid betracht. Door gebruik van de functie PLAUSIBILITY en reiniging van het communicatiekanaal door middel van de opdracht TERminate PRInt (= TERPRI) zodat de boodschap over kan komen die het conversatieprogramma ELIZA te bieden heeft.

Bij het programma dat vragen beantwoordt, hoort een programmaatje ELIZA, waarvan de tekst hier volgt:

```
(DEFUN ELIZA (NODE)

      ;;DOES THE OBVIOUS THING
      (PROG (XX NUM)
            (SETQ NUM (LENGTH (N NODE)))
            (RETURN
             (APPLY
              'APPEND
              (MAPLIST
               '(LAMBDA (WORD)
                        (COND ((NOT (LESSP NUM (LENGTH WORD))) NIL)   ;THIS KLUDGE STOPS IT AT THE END
                              ((SETQ XX (ASSQ (CAR WORD)
                                              '((I YOU) (ME YOU)
                                                (AM ARE) (ARE AM))))
                               (CDR XX))                              ;WE RETURN LIST OF THE THING READY
                              ((EQ (CAR WORD) 'YOU)                   ;THE APPLY APPEND CAN GET RID OF WORDS
                               (SETQ XX (FINDMOTHER WORD NODE))       ;UNFORTUNATELY, FOR 'YOU' IT IS TO
                               (COND ((ISQ XX SUBJ) '(I))             ;DECIDE WHETHER IT SHOULD BE 'I' OR
                                     ((ISQ XX OBJ) '(YOU))            ;'ME', ACCORDING TO WHETHER IT IS
                                     ((BUG ELIZA -- SUBJ OBJ))))      ;OBJECT OR SUBJECT. FINDMOTHER IS
                              ((LIST (CAR WORD)))))                   ;THE PARSE NODE. WORDS OTHER THAN
               (N NODE))))))                                          ;THESE ONES GO THROUGH DIRECTLY
```

VRAAG. Waarom heet de tekst ELIZA en wat doet het programmaatje?

3. Grapjes in de antwoorden:

VRAAG. Wat veroorzaken de volgende twee programmaatjes?

```
(DEFUN NAMESUGAR (NUM OSS)

      ;;GENERATES PHRASES LIKE 'THREE OF THEM'
      (PROG (VAGUE)
            (SETQ VAGUE (MEMQ   VAGUE (MARKERS? OSS)))                ;VAGUE IS FOR WORDS LIKE 'ANYTHING',
            (RETURN
             (LIST
              (CONS 'SAY                                              ;'SOMETHING', 'NOTHING' TO AVOID SAYING
                    (COND ((AND VAGUE (ZEROP NUM)) '(NOTHING))        ;THEM' WHEN IT ISN'T APPROPRIATE.
                          ((CONS (NAMENUM NUM)
                                 (COND (VAGUE (COND ((EQUAL NUM 1.)
                                                     '(THING))
                                                    ('(THINGS))))
                                       ('(OF THEM))))))))))
```

```
(DEFUN NOTELI NIL
       (GLOBAL-ERR THAT
                   ISN
                   'T
                   THE
                   KIND
                   OF
                   THING
                   I
                   CAN
                   BE
                   TOLD))
```

3.3.3.1 Inhoudsanalyse op basis van de systemic grammar

De grammatica's die we tot nu toe behandeld hebben maakten een scherpe scheiding tussen vorm (syntaxis) en inhoud (semantiek). Uit het voorgaande zal duidelijk zijn geworden dat een dergelijke scheiding niet gezond is wanneer men een inhoudsanalyse wil bedrijven van het soort dat bijvoorbeeld door Winograd werd gedaan.

Winograd moest dan ook op zoek gaan naar een ander soort grammatica. In tegenstelling tot de traditie zocht hij een taalkundige theorie die ervan uitgaat dat mensen taal altijd in de eerste plaats proberen te begrijpen. Die grammaticale theorie vond hij bij de Engelse linguist Halliday. Deze taalkundige ontwierp een theorie waarvan de basis is dat het onzinnig is taal te beschrijven zonder te focussen op betekenis. De analyse van de zin gaat in die theorie dan als volgt. De taalgebruiker hoort of leest woorden. Hij herkent direct of het gaat om structuurwoorden of om inhoudswoorden. Heeft hij te maken met een structuurwoord dan zoekt hij het erbij behorende inhoudswoord. Dit inhoudswoord brengt direct zijn betekenis mee. Met die betekenis wordt de analyse verrijkt. Nu zoekt de taalgebruiker weer verder met a l l e informatie die hij dan heeft.

Men zou ook als volgt kunnen redeneren. De taalgebruiker gebruikt alleen die informatie die hij strikt nodig heeft om mededelingen te begrijpen. Hij gebruikt alleen zinsontleding wanneer hij geen andere informatie tot zijn beschikking heeft. Zodra hij meer weet heeft hij de zinsontleding niet meer nodig. Dit systeem noemde Halliday 'systemic grammar'. Het lijkt enigszins op de deep case grammar, maar daar is de zin uitsluitend opgehangen aan de betekenis van het werkwoord. In de systemic grammar kan men ook met andere eenheden zoals nominale groepen al aardig uit de voeten. Uit psychologisch onderzoek is gebleken dat de nominale groep inderdaad een zeer belangrijke eenheid is bij taalverwerking, belangrijker dan de werkwoordgroep.

Winograd vertaalde de ideeën van de systemic grammar naar een computerprogramma toe. In dit programma hebben woorden (zie de woordenboekdefinities!) niet zomaar een statische b e t e k e n i s, nee, het zijn programma's, ze doen iets. Dat wil zeggen, komt een woord uit het woordenboek, dan verandert door de programma-opdrachten die in dat woord zitten de strategie van het programma.

Ofwel: de betekenissen van de woorden werken als een soort duiveltjes in een doosje. Ze worden ook wel DEMONS (duivels) genoemd. Dat deze theorie leidde tot een werkend systeem van inhoudsanalyse en taalverstaan kan men nog vandaag de dag gaan controleren in de Carnegie Mellon Universiteit, te Pittsburg, waar de robot staat die Winograd heeft bedacht.

3.3.3.2 Heeft u een vuurtje: Waar denken geen taal is

Stel dat u de vraag krijgt voorgelegd: Heeft u een vuurtje? Dan is het duidelijk dat bedoeld wordt: Heeft u een doosje lucifers waarin zich nog niet gebruikte lucifers bevinden en bent u bovendien bereid mij een niet gebruikte lucifer te doen benutten voor het aanmaken van een sigaret (sigaar, pijp, al naar gelang de toestand waarin deze vraag zich afspeelt).

Een taal-ontleedprogramma zou constateren:

1. Het gaat om een vraag.
2. Ik onderzoek of ik een vuurtje heb.
3. Ik kom tot de ontdekking dat mensen geen vuurtjes bij zich kunnen dragen.
4. Het programma antwoordt:
   "I'm not supposed to answer that kind of questions"
   (of een andere geïrriteerde opmerking).

Dat wil zeggen: er is aan het begrijpen van taal nog meer te verhapstukken dan de kennis van de wereld van objecten en het trekken van conclusies daaruit. Immers, een programma à la Winograd zou dan kennis moeten hebben van vuur, van mensen, van het gevaar van vuur en van de onzin die de vraag 'heb je een vuurtje' op grond daarvan in zich bergt.

Toch is de vraag naar een vuurtje erg gewoon. Deze vraag valt in de rij van vragen als:

- Kun je me zeggen waar Piet is?
  (antwoord van de taalmachine: ja, en verder niets)
- Begrijp je nu eindelijk hoe laat het is?
- De menukaart graag.
- Kunnen we even afrekenen?

en ga zo maar door.

Dit soort vragen vereist nog wat meer dan 'kennis van de wereld' in de trant van voorwerpen en de eigenschappen die daarbij horen. Het vereist kennis van de s i t u a t i e waarin een dergelijke vraag wordt gesteld. De situatie van het vuurtje is iemand die wil roken, maar geen lucifer (of aansteker!) bij zich heeft. De situatie van de vraag naar waar Piet is, is een situatie waarin iemand Piet nodig heeft en hem wil spreken of iets dergelijks. De situatie waarin de vraag voorkomt: "weet je nu eindelijk hoe laat het is", is een situatie waarin iemand aan een ander die naar zijn oordeel langzaam van begrip is, een complexe situatie probeert duidelijk te maken die de

langzame persoon persoonlijk aangaat. Het is niet in eerste instantie een vraag naar het vermogen van iemand om klok te kijken. De menukaart wordt gevraagd in een restaurant en heeft een hele serie implicaties: iemand die een menukaart vraagt, wil iets eten. Daarvoor moet hij echter iets bestellen. Dat kan alleen uit de voorraad die op de menukaart staat aangegeven. Na de bestelling verwacht hij dat hij bediend wordt, dat hij inderdaad krijgt wat hij besteld heeft. Waarop ook weer aan zijn kant de verplichting staat te betalen met of zonder fooi, al naar gelang het land waar men zich bevindt.

Kunnen we even afrekenen, zal men in eerste instantie ook rekenen bij de restaurant-situatie, ofschoon het ook kan horen bij een wraakoefening.

Nu zou men de kennis die wordt vereist om deze vragen te beantwoorden natuurlijk ook 'kennis van de wereld' kunnen noemen. We hebben 'kennis van de wereld' echter al gereserveerd voor een heel specifiek soort kennis. Namelijk de kennis van 'een en een is twee' of 'is x iets dat groter is dan iets anders, dan kan dat iets niet worden opgesloten in dat iets anders'.

Die kennis van de wereld zou men kunnen aanduiden met 'natuurkundig' of misschien ook 'wiskundig'. De kennis voor het vuurtje is kennis van situaties die per cultuur verschillend zijn of kunnen zijn. Dat is dus niet een voorraad van vaststaande kennis, maar kennis die slechts ten dele en onder zeer beperkende voorwaarden geldt. Kennis ook die men per situatie moet leren. Een aap kan wel leren dat een groot blok niet in een klein past, maar zich correct gedragen wanneer je hem vraagt: Hebt u een vuurtje, zal niet zo gemakkelijk te leren zijn voor een aap.

• SCRIPTS

De kennis waar het bij de vraag naar het vuurtje om gaat is kennis die cultureel bepaald is. Het is ook kennis met een serie open kaarten. Zo kan het vuurtje betekenen een lucifer, maar heeft de aangesprokene alleen maar een aansteker bij zich dan zal hij die aansteker presenteren, zonder dat iemand zich genomen voelt. Dit soort kennis is dus verzameld in een soort situatiebeschrijving waarin n o o d z a k e l i j k e maar ook m o g e l i j k e voorwaarden en gebeurtenissen/verwachtingen staan aangegeven. Noodzakelijk bij de vuurtjes-vraag is, dat er een instrument aankomt dat de sigaret/sigaar/pijp/ aan het branden brengt. Mogelijk is dat het gaat om een lucifer of een aansteker en eveneens dat het gaat om een sigaret, sigaar of een pijp. Het doet niet ter zake of de vraag gesteld wordt door een vrouw of een man. Ofschoon hierop restricties kunnen zitten, al naar gelang het milieu waarin men de vraag stelt, of de situatie waarin de vraag wordt gesteld.

Deze voorraad kennis zit in een soort schema dat met de Amerikaanse vakterm: SCRIPT wordt genoemd. De meeste teksten maken

gebruik van SCRIPTS. Een tekstmachine moet dus over scripts en de behandeling van scripts kunnen beschikken. Dat wil zeggen: gaat het over een vuurtje, dan moet de machine het 'sigaret-roken' script kunnen klaarzetten. Van daaruit worden de verwachtingen over wat er te doen en te gebeuren staat gestuurd. Hierdoor verdwijnen allerlei zuiver taalkundige meerduidigheden. Concreet: in de omgeving van het 'rook-script' zal het antwoord van de computer niet zijn: Ja (en verder niets) op de vraag: Hebt u een vuurtje? Neen, hij zal zijn aansteker tevoorschijn halen. Want hij heeft begrepen dat de vraag niet puur bedoeld is om de kennis van de vrager uit te breiden.

3.3.3.2.1 SAM PAM QUALM
De programmering van de SCRIPT-MACHINE

Uit het bovenstaande zal zijn duidelijk geworden dat niet in een paar bladzijden een computerprogramma kan worden gepresenteerd waarmee men zelf aan de gang kan gaan om een SCRIPT-MACHINE te programmeren. Nog ingewikkelder dan SHRDLU is dit programma immers. We moeten ons dus ook nu weer behelpen met een gedeelte output en een inzicht in de kern van de zaak. De kern van de zaak is de zogenaamde 'conceptual dependency theory' van Roger Schank. Met deze theorie en het programma SAM is het ook allemaal begonnen.

Eerst echter wat output van een programma dat verder gevorderd is dan SAM, namelijk het programma QUALM. Het programma QUALM zelf is weer samengesteld uit de programma's SAM en PAM. Aan al deze programma's is jaren gewerkt door Richard Cullingford, Gerald de Jong, Leila Habib, James Meehan, Richard Proutfoot, Christopher Riesbeck, Walter Stitzman en Robert Wilensky. En dat bovendien met de meest geavanceerde computers op de universiteit van Yale in het kader van het Yale Artificial Intelligence Project. Om te controleren of de machine inderdaad de tekst begrijpt, doet men het volgende. De computer krijgt een tekst te lezen. Na de leesfase moet hij vragen over de tekst beantwoorden. Als volgt:

Verhaaltje
Jan ging naar een restaurant en de kellner gaf hem een menu. Toen hij een hot dog bestelde, zei de serveerster dat ze er geen hadden. Dus bestelde Jan een hamburger. Maar toen de hamburger kwam was hij aangebrand en verliet Jan het restaurant.

Vragen (c = computer, m = mens)
m: Gaf de serveerster Jan de kaart?
c: NEE, DE KELLNER GAF HEM DE KAART
m: Bestelde Jan een hot dog?
c: JA
m: At Jan de hot dog op?
c: NEE, DE SERVEERSTER ZEI DAT ER GEEN HOT DOGS WAREN
m: Wat kreeg Jan geserveerd?

c: EEN HAMBURGER
m: Waarom at Jan die hamburger niet op?
c: OMDAT HIJ AANGEBRAND WAS
m: Was Jan kwaad?
c: IK DENK HET WEL
m: Waarom betaalde Jan niet?
c: JAN WAS KWAAD DAT ZIJN HAMBURGER AANGEBRAND WAS EN DAAROM GING HIJ WEG
m: Waarom at Jan geen hot dog?
c: OMDAT DE SERVEERSTER ZEI DAT ZE ER GEEN HADDEN
m: Wat gebeurde er toen Jan een hamburger bestelde?
c: DE SERVEERSTER GAF ZIJN BESTELLING DOOR AAN DE KOK, DE KOK MAAKTE DE HAMBURGER KLAAR. DE KOK GAF DE HAMBURGER AAN DE SERVEERSTER. DE SERVEERSTER SERVEERDE JAN DE HAMBURGER.
m: Wat gebeurde er toen Jan een hot dog bestelde?
c: DE SERVEERSTER VERTELDE JAN DAT ER GEEN HOT DOGS WAREN

Verhaaltje 2
Jan hield van Marie maar ze wilde niet met hem trouwen. Op een dag werd Marie door een draak van het kasteel geroofd. Jan sprong op zijn paard en doodde de draak. Marie stemde in een huwelijk toe. Ze leefden nog lang en gelukkig.

Vragen
m: Waarom sprong Jan op zijn paard?
c: OMDAT HIJ BIJ MARIE WILDE ZIJN
m: Waarom stemde Marie toe in een huwelijk?
c: OMDAT ZE ZICH VERPLICHT VOELDE
m: Waarom doodde Jan de draak?
c: OMDAT HIJ WILDE DAT MARIE BLEEF LEVEN

• Ketens van Gebeurtenissen (causal chains)

Als we nu de scripts hebben klaarstaan, moet er nog een lopend verhaal gemaakt kunnen worden waaruit allerlei niet expliciet genoemde informatie kan worden afgeleid. Daartoe moet de machine een keten van gebeurtenissen opbouwen. Die keten moet worden gevoegd door operatoren als MAAKT MOGELIJK, RESULTAAT, REDEN, BEGIN, KAN VEROORZAKEN, LEIDT TOT, enz. Dat wil zeggen: de machine maakt een 'causale keten representatie' van een verhaaltje. Dit is een stuk uit het begrijpen, het heeft met taal op zich niet zoveel te maken, het is een opbouwen van begrippen die bij elkaar horen en een aangeven hoe ze bij elkaar horen. Met een vreemd woord: conceptualisering van het verhaal.

Men kan het best een voorstelling krijgen van wat deze conceptualisering in causale kettingen betekent wanneer men bij een verhaaltje

denkt aan een film waarin men achter elkaar precies ziet wat er gebeurt. Als Jan naar een restaurant gaat neemt de camera op vanuit Jan. Gaan meer mensen naar een restaurant, dan komen er meer gezichtspunten en dus ook meer mogelijke causale ketenen. Om een voorbeeld te geven hoe het gaat geven we nu een verhaaltje met de causale keten die SAM ervan heeft gemaakt.

Jan ging met de bus naar New York. In de bus maakte hij een praatje met een oude dame. Bij het uitstappen bedankte hij de chauffeur. Hij nam de metro naar Leone. In de metro werd hij bestolen. Hij ging de trein uit en ging binnen bij Leone. Hij at wat lasagna. Toen de rekening kwam ontdekte hij dat hij niet kon betalen. De directie zei hem dat hij dan maar de borden moest gaan wassen.

- Causale keten der conceptualiseringen (men houdt nu eenmaal van woordgedrochten in de wetenschap)

Jan ging naar een bushalte. Hij wachtte een paar minuten. Hij ging de bus in. De chauffeur controleerde het kaartje van Jan. Hij liep naar een zitplaats. Hij ging erop zitten. Toen hij in de bus zat, sprak hij met een oude dame. De chauffeur reed Jan naar New York. Hij liep naar de chauffeur. Hij ging de bus uit. Hij liep een station in. Hij stak zijn kaartje in een automaat. Hij ging het perron op. Hij wachtte daar een paar minuten. Hij ging een coupé van de ondergrondse binnen. Een dief liep naar Jan. Hij haalde Jan's portefeuille uit zijn zak. De dief ging er vandoor. Jan liep naar een zitplaats. Hij ging erop zitten. De chauffeur bracht Jan naar Leone. Hij ging de metrocoupé uit. Hij ging het station uit. Hij ging binnen bij Leone. Hij keek binnen om zich heen. Hij zag dat er een tafel vrij was. Hij ging er naar toe. Hij ging op een stoel zitten. Hij bestelde lasagna. De ober zei tegen de chef dat Jan wilde dat hij iets voor hem klaarmaakte. De chef maakte de lasagna klaar. De ober kreeg die van de chef. De ober ging naar de tafel. Hij serveerde Jan de lasagna. Die at hem op. Hij had genoeg gegeten. Hij vroeg de ober om de rekening. Hij kreeg die van de ober. Jan las de rekening. Jan ontdekte dat hij de rekening niet kon betalen. Hij maakte de ober duidelijk dat hij niet kon betalen. De gerant zei Jan dat hij borden moest wassen. Hij ging de keuken in. Hij waste de borden. Hij verliet Leone.

Er zijn een paar dingen niet helemaal duidelijk in dit verhaal. We weten bijvoorbeeld niet wanneer precies de portefeuille gestolen werd voor of nadat Jan was gaan zitten. We weten zelfs niet zeker of hij wel gezeten heeft. Dat betekent: er zijn punten in de keten van gebeurtenissen die naar waarschijnlijkheid worden ingevuld.

In ieder geval moeten we om dit soort teksten te kunnen lezen een computerprogramma hebben dat verhalen herschrijft op de manier die boven aan de orde werd gesteld. Van die herschreven versie waarin geen stapjes meer ontbreken worden de antwoorden afgeleid op de

vragen die men naar het tekstbegrip stelt. Het is met name John Cullingford, die dit gedeelte van SAM heeft geschreven. Hiermee is een belangrijk deel van SAM beschreven. Op de bouwsteentjes die nog vóór de causale keten liggen komen we verderop terug bij de conceptual dependency theory van Schank.

- PAM: plannen maken

Om teksten te begrijpen is het niet alleen voldoende de keten van gebeurtenissen goed neer te zetten met een computerprogramma, er zijn ook verhaaltjes die men slechts kan begrijpen als men in staat is de motivatie van mensen te achterhalen. Anders gezegd: het programma moet kennis hebben over het waarom iemand iets doet. We hebben al een verhaaltje gehad waarin juist dit aspect heel belangrijk was, namelijk het verhaaltje van Jan, Marie en de draak. Het verhaal daarbij was dat Jan Marie van de draak redde en dat zij hem toen huwde. Dit heeft als verhaal alleen maar een zinnige betekenis als we zelf eerst afleiden dat de draak Marie iets wilde aandoen; dat Jan dat niet wilde; dat Marie dankbaar was tegenover Jan nadat hij haar gered had. Was het nu bijvoorbeeld zo dat Marie Jan huwde nadat hij haar lievelingspoedel aan een leeuw had gevoerd, dan zouden we te maken hebben met een totaal andere keten van motivaties. We zouden zelfs in eerste instantie bedenken dat we die reactie van Marie niet goed konden begrijpen. Slechts een cynisch ingestelde vrouwenhater zal bij zich zelf denken: Zie je wel, ik heb het altijd wel geweten dat ze zo zijn.

Deze uiteenzetting was bedoeld om duidelijk te maken dat er behalve rechttoe-rechtaan ketens van oorzaak en gevolg ook nog ketens mogelijk zijn waarbij men zoiets als kennis van menselijk gedrag en wat dat motiveert moet hebben. Dit te implementeren bleek zo moeilijk dat er een apart programma voor moest worden geschreven. Dat programma heet PAM. Dit programma maakt een causale keten, maar ook de plannen die gemaakt worden (causal chain en planning structure in de vaktaal). Behalve de gewone causale keten komt er door PAM nog informatie bij over het feit dat Marie Jan zo dankbaar was en dat ze hem daarom huwde. Ook dat er in Jan een gedachtenproces op gang kwam toen hij hoorde dat Marie gekidnapped was. Dit proces bracht hem tot het besluit haar te redden. Het is duidelijk dat dit soort oorzaak-en-gevolg-ketens de feitenketens uit het verhaal van de zakkenroller en Jan in het restaurant te boven gaan. Robert Wilensky programmeerde PAM.

- De Conceptual Dependency theorie: een kat zonder staart

In het voorgaande moest worden geconstateerd dat er bij het begrijpen van taal zich processen afspelen die los staan van taal. Taal geeft alleen maar een aanzet om de processen op gang te brengen. Het gaat daarbij om processen waardoor en waarmee kennis wordt verwerkt.

Wil men met een computer werken dan moet men een wijze van voorstellen (representeren) van die kennis vinden die programmeerbaar is. De vraag die zich dus direct aanbiedt is: hoe moet de kennis waar het hier nu om gaat worden gerepresenteerd?

In een normaal woordenboek wordt kennis gerepresenteerd in taal: het ene woord wordt met het andere woord gedefinieerd. Daardoor zijn woordenboeken altijd als katten die zich om vooruit te komen in hun eigen staart bijten. Ze voelen daarbij wel de pijn en de inspanning, maar ze hebben niet in de gaten dat ze niet vooruit komen, maar in een kring ronddraaien.

Is er een representatie van kennis mogelijk zonder taal? Volgens Roger Schank is die er, wanneer men uitgaat van een basis met denkatomen (conceptuele eenheden). De taaluiting moet nu op de eerste plaats in verband worden gebracht met die denkatomen. Verder moet men de regels zoeken die werken op die denkatomen. De werking van de regels samen met de atomen noemt men conceptualisering. Men ziet aan een paar voorbeeldzinnetjes wel in dat er iets voor te zeggen is, om betekenis en begrijpen los te maken van taal of liever van taalkunde. In het zinnetje:

De oude man ziet een dikke kat in zijn tuin

kan geen taalkundige operatie aangeven wie er nu in de tuin is, de kat of de oude man. Maar ook de volgende soort zinnetjes laten zien dat taalkundige middelen zo hun beperkingen hebben:

Piet's voorkeur voor Marie is gevaarlijk
Piet's voorraad bonen is eetbaar

Tweemaal staat er een zin die draait om het werkwoord 'is', en twee keer wordt er iets gezegd over iets dat aan Piet wordt toegekend. In de eerste zin is iets van Piet gevaarlijk, maar in de tweede zin zijn de bonen eetbaar en niet Piet. De taalkundige theorie staat voor enorme problemen wanneer ze tracht een sluitend geheel te bouwen voor bijvoorbeeld deze twee zinnen. De mens echter trekt zich van die theoretische moeilijkheden niets aan en schiet gewoon naar een ander niveau van informatieverwerking, waarin het zonneklaar is dat niet Piet, maar de bonen eetbaar zijn. De mens heeft al zijn kennis ter beschikking, niet alleen zijn taalkundige kennis. Roger Schank formaliseerde een gedeelte hiervan in zijn conceptual dependency theorie.

Hierin wordt een onderscheid gemaakt tussen primitieven (atomen) en relaties. De primitieven kunnen met elkaar in verband worden gebracht, zodat ze in het geheel van elkaar afhankelijk worden. Dit is de reden waarom de theorie de begripsafhankelijkheidstheorie (conceptual dependency theory) genoemd wordt.

Primitieve begrippen zijn:

PP: (picture producer) hetgeen men het best aanduidt met 'de handelende persoon'

ACT: (action) een eenheid die iets k a n doen

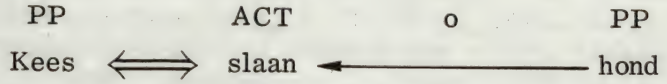
MODIFIER: deze brengen nuances aan, zowel op de PP, en dan zijn het PA's (picture aids), als aan de ACTs, en dan zijn het AA's (action aiders).

PP, ACT, PA en AA kunnen aan elkaar gerelateerd worden. Zo kan er tussen twee PP's een object relaties bestaan over een ACT. Voorbeeld:

Kees slaat zijn hondje

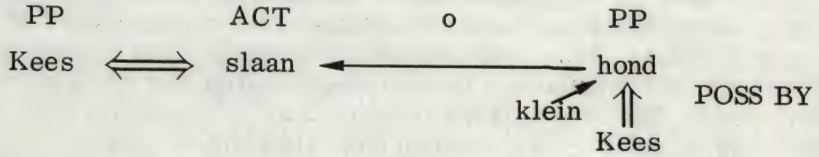
PP ACT PP

Het conceptuele diagram in de termen van Schank voor deze mededeling is:



Nu nog 'zijn' en '-je' inpassen:
'zijn' geeft een bezitsrelatie aan (Possessed by: POSS BY);
'-je' is een attribuut dat aangeeft dat de hond klein is.

Het complete diagram wordt nu:



VRAAG. Waar in het diagram zou de bepaling 'hard' terechtkomen bij de zin: Kees slaat zijn hondje hard?

Slaan veronderstelt dat iemand slaat en dat iets de slagen opvangt. Dit wordt weergegeven met visuele symbolen als pijlen. Boven de pijlen en in de vorm van de pijlen geeft men weer wat voor soort relaties het is. Vanuit deze soorten kan men nu precies voorspellen wat er gaat komen, en wat er nog nodig is. Zo vereist het begrip geven ook een ontvangen, of dat nu wordt uitgesproken of niet. Op deze manier is het programma in staat uit de kennisdefinities, waar immers ook de relaties in zitten, voorspellingen te doen, met name over de zaken die voor zich spreken. Dat wil zeggen de zaken die niet met zoveel woorden in de tekst staan.

Nu is althans het principe bekend waarop programma's als SAM en PAM aan de mogelijkheid komen een verhaaltje te herschrijven in

een causale ketting: de relaties die gelegd moeten worden tussen de atomen worden gewoon gelegd. Wat het oplevert, kan men bestuderen bij het herlezen van de uitgeschreven verhaaltjes op pagina 197.

3.3.3.2.2 Maanstenen, koeken bakken, reizen boeken:
LSNLIS, LUIGI en GUS: U vraagt, wij antwoorden

Met SHRDLU en QUALM is de hele overkapping gemaakt over de computerprogramma's heen die taal kunnen begrijpen. Deze overkapping hangt over systemen heen die nauw gebonden zijn aan een beperkt gebied van kennis (SHRDLU) tot systemen die heel algemeen zijn opgezet en die enkele basisfuncties van menselijke kennisverwerking nabootsen (QUALM). Daartussen in is nog een gebied van kennisverwerking en kennisoverdracht dat gebruik maakt van heel specifieke kennis van een bepaald gebied. Wilde Winograd uitzoeken hoe je taken die in het Engels zijn gesteld uitvoert met een computer, wilde de groep om Schank uitzoeken hoe menselijke geheugenprocessen in verband met kennisverwerking samenhangen, er zijn ook computerprogramma's die een soort encyclopedische taak hebben. Men kan met dergelijke programma's informatie opvragen uit bepaalde kennisgebieden. Deze programma's zijn geen van alle zo algemeen als het programma van Winograd en zeker niet als de programma's van de Schank-groep. Toch moeten ze genoemd worden, want het zijn de normale vraag/antwoord (question answering) programma's. De beroemdste daarvan is een programma van William Woods. Dit programma moest vragen beantwoorden over de analyse van materiaal dat was meegebracht van de maan na de eerste maanreis. Het programma heet LSNLIS. De opzet is als volgt. Eerst een ontleed-programma. Dit ontleed-programma wordt voorzien van betekenissen, krijgt dus een semantische interpretatie. Die semantische interpretatie is eenvoudig omdat het alleen maar gaat over maanstenen enz. De semantische processor is dus zeer beperkt, want het kan nergens anders over gaan. Vanaf die semantische interpretatie wordt via logische operaties het antwoord op de vraag gezocht. Zo wordt bijvoorbeeld de vraag gesteld:

Bevatten alle breciën lanthanum?
/breciën zijn steendelen, lanthanum is een stof/

Bij de oplossing van dit soort vraagstukken moet de computer een waarheidsvraag beantwoorden. Daarbij liggen twee methoden voor de hand. De eerste is de langzame en luie, namelijk: zoek in je data bestand alle breciën en kijk of er lanthanum in zit. Zo ja, dan is het antwoord: ja. Zo nee, dan is het antwoord dus: nee. LSNLIS gebruikt de luie methode.

Een andere methode is via een deductieproces, bijvoorbeeld als volgt. De computer weet dat alle stukken ieder scheikundig element bevatten. Breciën zijn stukken, lanthanum is een element; dus: alle breciën bevatten lanthanum. Op deze wijze kunnen alle vragen die in

dit soort informatiezoekende systemen aan de orde komen worden omgezet in zogenaamde proposities. Op deze proposities kan dan weer een logische berekening worden losgelaten. Deze logische berekening levert 'waar' of 'vals' op. Al naar gelang de uitkomst hiervan wordt de vraag met 'ja' of 'nee' beantwoord. De methode van Woods bevat dit element van logische berekening ('predicaat calculus') nog niet. Toch is Woods met zijn systeem heel beroemd geworden, omdat hij ermee de ATN heeft geïntroduceerd (zie paragraaf 3.3.3.1). Alle vraag/antwoord-systemen waaraan wiskundigen of informatici pur sang werken of gewerkt hebben, hebben dit kenmerk dat predicaatcalculus een fundamenteel stuk in het ontwerp is. Daarmee gaan de ontwerpers van deze systemen ervan uit dat zinnen in taal proposities zijn in de zin van de logica. Dat wil zeggen, dat zinnen het kenmerk hebben dat ze waar of onwaar zijn als het op de betekenis aankomt. Dit laatste nu geldt slechts voor een zeer klein gedeelte van ons taalgebruik. Als men de vraag moet beantwoorden: Waarom sprong Jan op zijn paard (zie het verhaaltje over de draak) of: Heeft u misschien een vuurtje, komt men niet ver met een berekening of de zin nu 'waar' is of 'onwaar'. Deze systemen passen hooguit in een omgeving waar alleen de feiten tellen. Feiten in de zin van aan uiterlijke eigenschappen vaststelbare gegevens. Of Jan van Marie houdt kan men op deze wijze niet berekenen en ook niet of u een vuurtje heeft. (Zie verder TALE SPIN, p. 221.)

Dit is dus weer het verhaal van de computerprogramma's die geschreven zijn voor een beperkt domein. Buiten dat domein zijn de ontworpen strategieën niet meer geldig. Dat is ook de reden waarom er zoveel van deze systemen zijn. Er is maar één reeks: SAM, PAM, QUALM, maar er zijn tientallen vraag/antwoord systemen voor beperkte domeinen. SAM, PAM, QUALM zijn zelfs toepasbaar op verschillende talen. Of nog mooier, men kan in het Engels de vraag stellen en zo men wil in het Nederlands antwoord krijgen. De vragen kunnen gaan over zaken die men door echt denken moet achterhalen. De andere vraag/antwoord systemen hebben doorgaans een beperkte woordenschat, vooral in hun antwoordenrepertoire. Ze zijn zeker niet onafhankelijk van de taal waarin het vraag/antwoord systeem is opgezet.

- LUIGI/GUS

Het programma LUIGI kan antwoord geven op de vraag:

Wat heb ik nodig om brood te bakken?
of: Hoe moet je koekjes bakken?

Het programma LUIGI werd gemaakt door Scragg. Het bevat een soort scripts, maar dan voor de keuken. Het kan ook alleen maar processen beschrijven.

Een redelijk interessant programma is het programma GUS. Met

dit programma kan men via een dialoog met de computer een vliegreis boeken. Het programma bestaat uit een serie vakken waar een specifieke informatie in terecht moet komen (zogenaamde frames). Het programma probeert die vakken te vullen met de vereiste informatie. Zijn de vakken vol, dan is de reis geboekt. Het interessante aan GUS is dat het zelf de vragen stelt en probeert de antwoorden te begrijpen om weer een nieuwe vraag te kunnen stellen, terwijl de normale vraag/antwoord systemen zelf de vragen moeten beantwoorden, ze stellen ze niet.

GUS werd gemaakt, of liever gezegd er wordt nog aan gewerkt, door Bobrow en Winograd. De meeste aandacht wordt besteed aan het opstellen van de frames. Een dialoog met GUS ziet er als volgt uit:

GUS:	Hello. My name is GUS. I can help you plan a simple trip by air. Where do you want to go?
(1) Client:	I want to go to San Diego on May 28
GUS:	What time do you want to leave?
(2) Client:	I must be in San Diego before 10 am
GUS:	Would you like P.S.A. flight 102 that arrives at 9.15 am?
(3) Client:	What is the next flight?
GUS:	Air California flight 310 that leaves at 8.30 am and arrives at 10.00 am
(4) Client:	I'll take the first one
GUS:	In what name should I make a reservation, please?
(5) Client:	Dan Bobrow
GUS:	I have confirmed the following flight: P.S.A. flight 102 on Wednesday May 28 from San Jose to San Diego that leaves at 7.30 am and arrives at 9.15 am. What date do you want to return on?
(6) Client:	On Friday in the evening
GUS:	Would you like the flight that leaves at 7.45 pm?
(7) Client:	That's fine
GUS:	I have confirmed the following flight: P.S.A. flight 307 on Friday May 30 from San Diego to San Jose that leaves at 7.45 pm and arrives at 9.30 pm. Thank you for calling. Goodbye.

Uit: Artificial Intelligence 8 (1977), 155-173.

3.3.3.3 Automatisch vertalen, 2

In paragraaf 2.5 is automatisch vertalen al eenmaal aan de orde geweest. We konden toen niet verder komen dan het zogenaamde woord-voor-woord vertalen, wat helemaal niets met vertalen te maken bleek te hebben. Op grond van het tot nu toe behandelde is al duidelijker geworden wat nodig is voor automatisch vertalen. Om

automatisch te kunnen vertalen moet de machine beschikken over de mogelijkheid een representatie van de betekenis van een tekst te maken. Deze representatie moet onafhankelijk zijn van welke taal (natuurlijke taal) dan ook. Heeft men een dergelijke representatie, dan hoeft men slechts de regels in te voeren die gelden voor de formulering van zinnen in een bepaalde taal. Men zou nu dus direct kunnen denken aan de representaties die kunnen worden gemaakt met de conceptual dependency theorie. Men zou dus dat gedeelte van SAM uit SAM kunnen lichten en dan de regels schrijven voor Nederlandse, Franse, Chinese zinnen. Doet men dat dan heeft men een vertaalmachine vanuit het Engels naar die andere talen.

Er is echter nog een ander, eenvoudiger systeem bedacht, en wel door Yorrick Alexander Wilks. Wilks bedacht ook een serie semantische primitieven. In zijn BNF notatie zien die er als volgt uit.

```
(*AL) ::= <DTHIS|THIS|MAN|FOLK|GRAIN|PART|WORLD|STUFF|THING|
          BEAST|PLANT|SPREAD|LINE|ACT|STATE>
          (*AL means all substantive elements)

(*EN) ::= <DTHIS|THIS|MAN|FOLK|GRAIN|PART|STUFF|THING|BEAST|
          PLANT|SPREAD|LINE>
          (*EN means elements that are entities)

(*AN) ::= <MAN|FOLK|BEAST|GRAIN>
          (*AN means animate antities. GRAIN is used as the main
          element for social organizations, like the Red Cross)

(*PO) ::= <DTHIS|THIS|MAN|FOLK|GRAIN|PART|STUFF|THING|ACT|BEAST|
          PLANT|STATE>
          (*PO means potent elements, those that can designate
          actors. The class cannot be restricted to *AN since
          rain wets the ground, and the wind opens doors)

(*SO) ::= <STUFF|PART|GRAIN|SPREAD>

(*MA) ::= <ACT|SIGN|STATE>
          (*MA designates mark elements, those that can
          designate items that themselves designate like
          thoughts and writings)
```

Woorden worden onmiddellijk omgezet in hun semantische primitieven. De machine verzamelt de bij elkaar behorende informatie op grond van die primitieven, dat wil zeggen alles wat gecombineerd kan worden, wordt samengebracht. Kunnen semantische primitieven op meer dan een manier worden verbonden dan maakt de machine twee zogenaamde templates. Zijn alle mogelijke templates van een tekst gemaakt, dan gaat de machine na welke template de meeste

verbindingen tussen de semantische primitieven heeft. Anders gezegd: welke template het meest compleet geschakeld is. De meest complete schakeling wordt gekozen als de beste representatie van de tekst. Op die manier worden de normale meerduidigheiden van woorden uitgeschakeld. Is de representatie van de betekenis gevonden, dan wordt gezocht wat de betekeniselementen daarin in de andere taal betekenen. Daarna worden de zinnen in de andere taal geformuleerd op de wijze waarop dat in die andere taal hoort.

Wat aan het systeem van Wilks het meest opvalt is, dat de machine eigenlijk niet hoeft te begrijpen wat de tekst betekent die hij vertaalt. Hij gaat gewoon uit van de voor de hand liggende instelling: waar de meeste verbindingen worden gemaakt, daar moet ik zijn bij mijn keuze in het geval van meerduidigheid. Dat is een voor de hand liggende reden: waar het meeste klopt, dat zal ook wel het juiste zijn.

Dit systeem had een enorm voordeel. Hiermee kan men namelijk ook b e e l d s p r a a k oplossen. Het systeem is niet afhankelijk van vooraf opgegeven kennis zoals scripts of frames, neen, het systeem ademt met de tekst mee. Als men nu aan het eind van een verhandeling over het openen van deuren spreekt over een slot, dan weet de machine op grond van de informatiedichtheid waarvan de betekenis 'deurslot' deel uitmaakt, dat hij die betekenis moet kiezen en niet de betekenis 'kasteel' of 'de slotmaten van een symfonie'. Ging de verhandeling over kastelen, dan is duidelijk dat de machine kiest: 'kasteel' voor slot. Het raadsel dat in het systeem van Wilks zit is, dat de machine niet begrijpt wat er gezegd wordt, maar dat hij niettemin de juiste vertaling oplevert.

De volgende output is van het programma van Wilks. Het is de output van een Engelse tekst die in het Frans werd vertaald.

```
(DURING THE WAR CM)
(PENDANT LA GUERRE VG)
[DURING:GAVEUP: location:∅:DTHIS PBE ACT]

(HITLER GAVE UP THE EVENING SHOWINGS CM)
(HITLER RENONCA AUX REPRESENTATIONS DU SOIR VG)
[nil:nil:nil:∅:MAN DROP ACT]

(SAYING)
(DISANT)
[nil:HITLER:nil:3:DTHIS DO DTHIS]

(THAT HE WANTED)
(QU'IL VOULAIT)
[THAT:SAYING:object: 3:MAN WANT DTHIS]

(TO RENOUNCE HIS FAVORITE ENTERTAINMENT)
(RENONCER A SA DISTRACTION FAVORITE)
[TO:WANT:object:3:DTHIS DROP ACT]

(OUTOF SYMPATHY)
(PAR SYMPATHIE)
[OUTOF:RENOUNCE:source:3:DTHIS PDO SIGN]
```

(FOR THE PRIVATIONS OF THE SOLDIERS PD)
(POUR LES PRIVATIONS DES SOLDATS PT)
[FOR:SYMPATHY:recipient:3:DTHIS PBE ACT]

(INSTEAD RECORDS WERE PLAYED PD)
(A LA PLACE ON PASSA DES DISQUES PT)
[INSTEAD:nil:nil:Ø:MAN USE THING](comment:template is active)

(BUT)
(MAIS)
[BUT:nil:nil:Ø:No Template]

(ALTHOUGH THE RECORD COLLECTION WAS EXCELLENT CM)
(BIEN QUE LA COLLECTION DE DISQUES FUT EXCELLENTE VG)
[ALTHOUGH:PREFERRED:nil:Ø:GRAIN BE KIND]

(HITLER ALWAYS PREFERRED THE SAME MUSIC PD)
(HILTER PREFERAIT TOUJOURS LA MEME MUSIQUE PT)
[nil:nil:nil:Ø:MAN WANT GRAIN]

(NEITHER BAROQUE)
(NI LA MUSIQUE BAROQUE)
[NEITHER MUSIC:qualifier:Ø:DTHIS DBE KIND]

(NOR CLASSICAL MUSIC CM)
(NI CLASSIQUE VG)
[NOR INTERESTED:nil:Ø:GRAIN DBE DTHIS]

(NEITHER CHAMBER MUSIC)
(NI LA MUSIQUE DE CHAMBRE)
[NEITHER INTERESTED:nil:Ø:GRAIN DBE DTHIS]

(NOR SYMPHONIES)
(NI LES SYMPHONIES)
[NOR INTERESTED:nil:Ø:GRAIN DBE DTHIS]

(INTERESTED HIM PD)
(NE L'INTERESSAIENT PT)
[nil:nil:nil:1:DTHIS CHANGE MAN]

(BEFORELONG THE ORDER OF THE RECORDS BECAME VIRTUALLY FIXED PD)
(BIENTOT L'ORDRE DES DISQUES DEVINT VIRTUELLEMENT FIXE PT)
[BEFORELONG:nil:nil:Ø:GRAIN BE KIND]

(FIRST HE WANTED A FEW BRAVURA SELECTIONS)
(D'ABORD IL VOULAIT QUELQUES SELECTIONS DE BRAVOURE)
[nil:nil:nil:Ø:MAN WANT PART]

(FROM WAGNERIAN OPERAS CM)
(D'OPERAS WAGNERIENS VG)
[FROM:SELECTIONS:source:3:DTHIS PDO GRAIN]

(TO BE FOLLOWED PROMPTLY)
(QUI DEVAIENT ETRE SUIVIES RAPIDEMENT)
[TO:OPERAS:nil:3:MAN DO DTHIS](comment: shift to active
template again may give a different but not incorrect
translation)

(WITH OPERETTAS PD)
(PAR DES OPERETTAS)
[WITH:FOLLOWED:nil:3:DTHIS PBE GRAIN]

```
(THAT REMAINED THE PATTERN PD)
(CELA DEVINT LA REGLE PT)
[nil:nil:nil:Ø:THAT BE GRAIN](comment:no mark because 'that'
ties to a whole sentence.)

(HITLER MADE A POINT OF TRYING)
(HITLER SE FAISAIT UNE REGLE D'ESSAYER)
[nil:nil:nil:Ø:MAN DO DTHIS]

(TO GUESS THE NAMES OF THE SOPRANES)
((DE DEVINER LES NOMS DES SOPRANOS)
[TO:TRYING:object:2:DTHIS DO SIGN]

(AND WAS PLEASED)
(ET ETAIT CONTENT)
[AND:HITLER:nil:3:DTHIS BE KIND]

(WHEN HE GUESSED RIGHT CM)
(QUAND IL DEVINAIT JUSTE VG)
[WHEN:PLEASED:location:3:MAN DO DTHIS]

(AS HE FREQUENTLY DID PD)
(COMME IL LE FAISAIT FREQUEMMENT PT)
[AS:GUESSED:manner:3:MAN DO DTHIS]
```

Gedeelten uit het programma van Wilks:

```
(DEFPROP CURR
   (LAMBDA (L)
   (PROG (Q X R U)
    (SETQ U (CDR L))
T1  (COND ( (NULL U)
            (RETURN (CONS NIL L))))
    (SETQ X (CAR U))
    (COND ( (PUNC X)
            (RETURN (CONS NIL L))))
    (COND ( (MEMBER X WDLIS)
            (RETURN ( CONS NIL L))))
    (COND ( (AND ( PRINCV X)
                 (NOT (AUX X)))
            (GO T2)
    (COND ( (AND (AUX X)
                 (NOT (EQUAL (CADR U)
                             (QUOTE NOT)))
                 (NOT (PRINCV (CADR U))))
            (PROG2
            (SETQ R  (QUOTE OKAUX))
    (COND (  (AND (PRINCN X)
                  (OR (PRINCV (CADR U))
                      (PRINCA (CADR U))
                      (PRINCADV (CADR U)))
                  (NOT (NULL Q))
                  (NOT (PREP (CADR U))))
                            (RETURN (CONS
                                      (SAG L (CDR U))
                                      (CDR U)))))
```

```
T3 (SETQ U (CDR U))
   (GO T1)



   (SETQ Q (QUOTE VERB))
   (COND ( (AND
              (OR (PRINCADV (CADR U))
                  (NOT (NULL R))
                  (MEMBER (CADR U)
                          (GET X (QUOTE PREPTI))
              (OR (PUNC (CADDR U))
                  (MEMBER (CADDR U) WDLIS)
                  (PRINCV (CADDR U)))))
                  (RETURN (CONS (SAG L
                                     (CDDR U))
                                (CDDR U ))))))

   (COND ( (OR (PUNC (CADR U))
               (AND  (NULL R)
                     (PRINCA (CADR U))
               (MEMBER (CADR U) WDLIS)
               (PRINCV (CADR U)))
          (RETURN (CONS (SAG L
                        (CDR U))
                        (CDR U)))))

   (GO T3)
   (SETQ Y (CDR U))
   (GO T1)
          .
          .
T2        .
          .
          .
          .
         etc.
```

- Niet psychologisch of taalkundig gefundeerde vraag-antwoord-systemen

De vraag/antwoord-systemen die in deze paragraaf werden behandeld hadden allemaal wel een taalkundige component. Het minst taalkundig georiënteerd zijn de systemen die ontwikkeld werden door de groep om Schank. Toch blijft de taal ook daar een niet te verwaarlozen centrale rol spelen. Nog het minst taalkundig georiënteerd zijn de systemen die proberen de input in natuurlijke taal zo snel mogelijk om te zetten in een vaste vorm (een zogenaamde canonieke vorm) ten behoeve van logische berekeningen. Deze systemen worden gebruikt als het gaat om encyclopedische kennisvragen die moeten worden afgeleid uit een data-bestand. In deze systemen ziet men, zoals wiskundigen meestal geneigd zijn, taal en taalkunde als een lastige bijkomstigheid. Een lastenbron die zo snel mogelijk uitgeschakeld moet worden zodat de onderzoeker dan weer in de Elyzeïsche velden van de wiskunde terecht kan komen. Daar immers bestaan geen meerduidigheden.

Jammer genoeg vergeet men bij deze redenering dat er misschien inderdaad wel terreinen in de wiskunde zijn waar geen of nauwelijks meerduidigheden zijn, maar dat dat terrein dan ook prompt over bijna niets meer kan gaan (geen inhoud meer heeft). Immers, het ideaal van volstrekte eenduidigheid kan alleen bestaan wanneer men voor iedere betekenis één enkele representatie heeft. Dan echter zou taal gelijk worden aan de werkelijkheid zelf. Een streven dat men beter kan vergeten. Wanneer taal immers identiek zou worden aan de werkelijkheid waarover ze communiceert, is een van de twee overbodig, en zou alle moeite die de mens doet om zijn taal te leren verspilde energie zijn. Er wordt weliswaar zeer veel energie verspild, maar niet op een dergelijk niveau.

Energie wordt verspild, zodra men systemen wenst te bedenken die beter zijn dan de systemen waarover de mens intuïtief beschikt. Wiskunde bijvoorbeeld als vervanger van natuurlijke taal is zo'n verspilling. Wiskunde op haar eigen terrein is echter geen verspilling. Er zijn in taal gebieden die wiskundig zijn. Het is goed de wiskunde daar dan ook in te schakelen. Prachtige voorbeelden hiervan vormen de moderne EXPERT systemen. Voor een boeiend overzicht hiervan zie het artikel van Duda en Gasching in Byte van september 1981, waarin ook een BASIC programma staat van een expert-systeem over dieren dat Winston en Horn in hun nieuwe LISP-boek oplepelen.

• STUDENT: Het algebra probleem

Een van de eerste vraag/antwoord systemen, en tevens een van de meest beroemde, is het programma STUDENT van Daniel Bobrow, een naam die we in dit boek al meer zijn tegengekomen. Bobrow is bijvoorbeeld de uitvinder van METEOR. METEOR werd ontwikkeld voor STUDENT.

Het programma STUDENT werd geschreven om zogenaamde ingeklede vergelijkingen die in het Engels werden gesteld op te lossen. Voorbeeld (output van STUDENT):

```
(THE PROBLEM TO BE SOLVED IS)
(THE SUM OF THREE NUMBERS IS 100. THE THIRD NUMBER EQUALS THE
SUM OF THE FIRST TWO NUMBERS. THE DIFFERENCE BETWEEN THE FIRST
TWO NUMBERS IS 10 PER CENT OF THE THIRD NUMBER. FIND THE THREE
NUMBERS.)

(THE EQUATIONS TO BE SOLVED ARE)
(EQUAL GO2536 (THIRD NUMBER))
(EQUAL GO2535 (SECOND NUMBER))
(EQUAL GO2535 (SECOND NUMBER))
(EQUAL GO2534 (FIRST NUMBER))
(EQUAL (PLUS (FIRST NUMBER) (MINUS (SECOND NUMBER))) (TIMES
.1000 (THIRD NUMBER)))
(EQUAL (PLUS (FIRST NUMBER) (PLUS (SECOND NUMBER) (THIRD
NUMBER))) 100)
```

```
(THE FIRST NUMBER IS 27.50)
(THE SECOND NUMBER IS 22.50)
(THE THIRD NUMBER IS 50)

(THE PROBLEM TO BE SOLVED IS)
(IF C EQUALS B TIMES D PLUS 1, AND B PLUS D EQUALS 3, AND B
MINUS D EQUALS 1, FIND C.)

(THE EQUATIONS TO BE SOLVED ARE)
(EQUAL G02539 (C))
(EQUAL (PLUS (B) (MINUS (D))) 1)
(EQUAL (PLUS (B) (D)) 3)
(EQUAL (C) (PLUS (TIMES (B) (D)) 1)
```

Zie verder de voorbeelden op pp. 211 en 212.

- Hoe zit STUDENT in elkaar?

De eerste stap van STUDENT is: lees het probleem en druk het af.
De tweede stap is: (substitueer) Vervang standaard strings door de standaard strings die STUDENT gebruikt.
Voorbeeld: komt in de tekst voor: IS EQUAL TO dan vervangt de patroonherkenner dat door IS.
Komt in de tekst voor: YEARS YOUNGER THAN dan vervangt de patroonherkenner dat door LESS THAN.
YEARS OLDER THAN wordt vervangen door PLUS
MORE THAN wordt ook vervangen door PLUS
enzovoort.

Daarna wordt een woordenboek geraadpleegd. In dit woordenboek wordt aan de woorden een eigenschap toegekend. Deze eigenschappen zijn vooral informatie over wat voor soort operator een bepaald woord inhoudt. Bijvoorbeeld:

De zin: DISTANCE EQUALS SPEED TIMES TIME
wordt omgezet in:

(EQUAL(DISTANCE) (TIMES(SPEED)TIME)))

Dit wordt omgezet in een serie statements van de volgende vorm:

P1 IS AN OPERATOR OF LEVEL k

concreet: TIMES IS AN OPERATOR OF LEVEL 1.
Het woordenboek zorgt er nu voor dat TIMES krijgt toegekend OP en k; ofwel TIMES ziet er na het woordenboek zo uit: (TIMES/OP 1). Zo wordt ieder woord voorzien van de erbij behorende informatie die moet dienen om het probleem op te lossen.

Het is duidelijk dat deze manier van doen veel lijkt op de manier waarop het zinsontledingsprobleem met METEOR werd opgelost. Alleen is nu de informatie in de property list niet taalkundig, maar algebraïsch van aard.

```
(THE PROBLEM TO BE SOLVED IS)
(IF THE NUMBER OF CUSTOMERS TOM GETS IS TWICE THE SQUARE OF
20 PER CENT OF THE NUMBER OF ADVERTISEMENTS HE RUNS , AND THE
NUMBER OF ADVERTISEMENTS HE RUNS IS 45 , WHAT IS THE NUMBER
OF CUSTOMERS TOM GETS Q.)

(WITH MANDATORY SUBSTITUTIONS THE PROBLEM IS)
(IF THE NUMBER OF CUSTOMERS TOM GETS IS 2 TIMES THE SQUARE
20 PERCENT OF THE NUMBER OF ADVERTISEMENTS HE RUNS , AND THE
NUMBER OF ADVERTISEMENTS HE RUNS IS 45 , WHAT IS THE NUMBER
OF CUSTOMERS TOM GETS Q.)

(WITH WORDS TAGGED BY FUNCTION THE PROBLEM IS)
(IF THE NUMBER (OF / OF) CUSTOMERS TOM (GETS / VERB) IS 2 (
TIMES / OP 1) THE (SQUARE / OP 1) 20 (PERCENT / OP 2) (OF /
OP) THE NUMBER (OF / OP) ADVERTISEMENTS (HE / PRO) RUNS , AND
THE NUMBER (OF / OP) ADVERTISEMENTS (HE / PRO) RUNS IS 45 ,
(WHAT / QWORD) IS THE NUMBER (OF / OP) CUSTOMERS TOME (GETS
/ VERB) (QMARK / DLM))


(THE SIMPLE SENTENCES ARE)

(THE NUMBER (OF / OP) CUSTOMERS TOM (GETS / VERB) IS 2 (TIMES
/ OP 1) THE (SQUARE / OP 1) 20 (PERCENT / OP 2) (OF / OP) THE
NUMBER (OF / OP) ADVERTISEMENTS (HE / PRO) RUNS (PERIOD / DLM))

(THE NUMBER (OF / OP) ADVERTISEMENTS (HE / PRO) RUNS IS 45
(PERIOD / DLM))

((WHAT / QWORD) IS THE NUMBER (OF / OP) CUSTOMERS TOM (GETS
/ VERB) (QMARK / DLM))


(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL G02515 (NUMBER OF CUSTOMERS TOM (GETS / VERB)))

(EQUAL (NUMBER OF ADVERTISEMENTS (HE / PRO) RUNS) 45)

(EQUAL (NUMBER OF CUSTOMERS TOM (GETS / VERB)) (TIMES 2 (EXPT
(TIMES  .2000 (NUMBER OF ADVERTISEMENTS (HE / PRO) RUNS)) 2)))


(THE NUMBER OF CUSTOMERS TOM GETS IS 162)
```

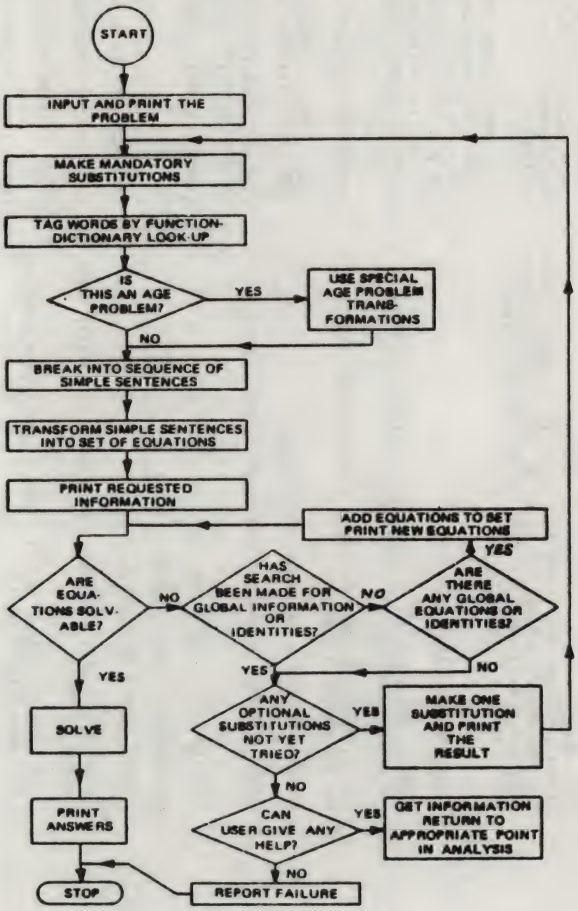
```
(THE PROBLEM TO BE SOLVED IS)
(THE SUM OF LOIS SHARE OF SOME MONEY AND BOB S SHARE IS $  4.500
. LOIS SHARE IS TWICE BOB s . FIND BOB S AND LOIS SHARE .)

(WITH MANDATORY SUBSTITUTIONS THE PROBLEM IS)
(SUM LOIS SHARE OF SOME MONEY AND BOB S SHARE IS  4.500 DOLLARS
. LOIS SHARE IS 2 TIMES BOB S . FIND BOB S AND LOIS SHARE .)

(WITH WORDS TAGGED BY FUNCTION THE PROBLEM IS)
((SUM / OP) LOIS SHARE (OF / OP) SOME MONEY AND BOB S SHARE
IS  4.500 DOLLARS (PERIOD / DLM) LOIS SHARE IS 2 (TIMES / OP
1) BOB S (PERIOD / DLM) (FIND / QWORD) BOB S AND LOIS SHARE
(PERIOD / DLM))


(THE SIMPLE SENTENCES ARE)

((SUM / OF) LOIS SHARE (OF / OP) SOME MONEY AND BOB S SHARE
IS  4.500 DOLLARS (PERIOD / DLM))

(LOIS SHARE IS 2 (TIMES / OP 1) BOB S (PERIOD / DLM))

((FIND / QWORD) BOB S AND LOIS SHARE (PERIOD / DLM))


(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL G02519 (LOIS SHARE))

(EQUAL G02511 (BOB S))

(EQUAL (LOIS SHARE) (TIMES 2 (BOB S)))

(EQUAL (PLUS (LOIS SHARE OF SOME MONEY) (BOB S SHARE)) (TIMES
 4.500 (DOLLARS)))


THE EQUATIONS WERE INSUFFICIENT TO FIND A SOLUTION

(ASSUMING THAT)
((BOB S) IS EQUAL TO ( BOB S SHARE))

(ASSUMING THAT)
((LOIS SHARE) IS EQUAL TO (LOIS SHARE OF SOME MONEY))


(BOB S IS  1.500 DOLLARS)

(LOIS SHARE IS  3 DOLLARS)
```

```
(THE PROBLEM TO BE SOLVED IS)
(MARY IS TWICE AS OLD AS ANN WAS WHEN MARY WAS AS OLD AS ANN
IS NOW . IF MARY IS 24 YEARS OLD , HOW OLD IS ANN Q.)

(WITH MANDATORY SUBSTITUTIONS THE PROBLEM IS)
(MARY IS 2 TIMES AS OLD AS ANN WAS WHEN MARY WAS AS OLD AS
ANN IS NOW . IF MARY IS 24 YEARS OLD , WHAT IS ANN Q.)

(WITH WORDS TAGGED BY FUNCTION THE PROBLEM IS)
((MARY / PERSON) IS 2 (TIMES / OP 2) AS OLD AS (ANN / PERSON)
WAS WHEN (MARY / PERSON) WAS AS OLD AS (ANN / PERSON) IS NOW
(PERIOD / DLM) IF (MARY / PERSON) IS 24 YEARS OLD , (WHAT /
QWORD) IS (ANN / PERSON) (QMARK / DLM))


(THE SIMPLE SENTENCES ARE)

((MARY / PERSON) S AGE IS 2 (TIMES / OP 1) (ANN / PERSON) S
AGE G02521 YEARS AGO (PERIOD / DLM))

(G02521 YEARS AGO (MARY / PERSON) S AGE IS (ANN / PERSON) S
AGE NOW (PERIOD / DLM))

((MARY / PERSON) S AGE IS 24 (PERIOD / DLM))

((WHAT / QWORD) IS (ANN / PERSON) S AGE (QMARK / DLM))


(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL G02522 ((ANN / PERSON) S AGE))

(EQUAL ((MARY / PERSON) S AGE) 24)

(EQUAL (PLUS ((MARY / PERSON) S AGE) (MINUS (G02521))) ((ANN
/ PERSON) S AGE))

(EQUAL ((MARY / PERSON) S AGE) (TIMES 2 (PLUS ((ANN / PERSON)
S AGE) (MINUS (G02521)))))


(ANN S AGE IS  18)
```

```
(THE PROBLEM TO BE SOLVED IS)
(THE SUM OF THE PERIMETER OF A RECTANGLE AND THE PERIMETER
OF A TRIANGLE IS 24 INCHES . IF THE PERIMETER OF THE RECTANGLE
IS TWICE THE PERIMETER OF THE TRIANGLE , WHAT IS THE PERIMETER
OF THE TRIANGLE Q.)

(WITH MANDATORY SUBSTITUTIONS THE PROBLEM IS)
(SUM THE PERIMETER OF A RECTANGLE AND THE PERIMETER OF A TRIANGLE
IS 24 INCHES . IF THE PERIMETER OF THE RECTANGLE IS 2 TIMES
THE PERIMETER OF THE TRIANGLE , WHAT IS THE PERIMETER OF THE
TRIANGLE Q.)

(WITH WORDS TAGGED BY FUNCTION THE PROBLEM IS)
((SUM / OP) THE PERIMETER (OF / OP) A RECTANGLE AND THE PERIMETER
(OF / OP) A TRIANGLE IS 24 INCHES (PERIOD / DLM) IF THE PERIMETER
(OF / OP) THE RECTANGLE IS 2 (TIMES / OP 1) THE PERIMETER (
OF / OP) THE TRIANGLE , (WHAT / QWORD) IS THE PERIMETER (OF
/ OP) THE TRIANGLE (QMARK / DLM))


(THE SIMPLE SENTENCES ARE)

((SUM / OP) THE PERIMETER (OF / OP) A RECTANGLE AND THE PERIMETER
(OF / OP) A TRIANGLE IS 24 INCHES (PERIOD / DLM))

(THE PERIMETER (OF / OP) THE RECTANGLE IS 2 (TIMES / OP 1)
THE PERIMETER (OF / OP) THE TRIANGLE (PERIOD / DLM))

((WHAT / QWORD) IS THE PERIMETER (OF / OP) THE TRIANGLE (QMARK
/ DLM))


(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL G02517 (PERIMETER OF TRIANGLE))

(EQUAL (PERIMETER OF RECTANGLE) (TIMES 2 (PERIMETER OF TRIANGLE)))

(EQUAL (PLUS (PERIMETER OF RECTANGLE) (PERIMETER OF TRIANGLE))
(TIMES 24 (INCHES)))


(THE PERIMETER OF THE TRIANGLE IS  3 INCHES)
```

Als alle woorden zijn voorzien van hun betekenissen voor de taak die moet worden opgelost, worden alle zinnen opgesplitst in een serie simpelere zinnen. Samengestelde zinnen bestaan dus niet meer. Nu worden enkelvoudige zinnetjes omgezet in de algebraïsche representatie van de ingeklede vergelijking. Aan de hand van de twee voorbeeldjes op p. 206/207 is deze stap goed te verifiëren. Daarna wordt een eenvoudig rekenprogramma aangeroepen dat de vergelijkingen oplost. Mochten de vergelijkingen in dit stadium nog niet oplosbaar zijn, dan wordt opnieuw het programma aangeroepen dat substitueert (stap 2). Daarin wordt bijvoorbeeld geprobeerd of er nog andere substituties mogelijk waren met de tekst dan de veranderingen die nu werden aangebracht. Lukt dat niet dan vraagt het programma zelf om hulp aan de gebruiker. Kan de gebruiker geen informatie geven, dan wordt de mislukking gemeld.

Het blokschema van STUDENT is als volgt (uit Minsky, 1968, p. 206).



- Een stukje STUDENT

```
(* ($)
   (1 (FN TERPRI)
      (FN TERPRI)
      (FN TERPRI))
   *)
(* ((*P The problem to be solved is:))
   *)
(IDIOMS ($)
        *)
(* (HOW OLD)
   (WHAT)
   IDIOMS)
(* (IS EQUAL TO)
   (IS)
   IDIOMS)
(* (YEARS YOUNGER THAN)
   (LESS THAN)
   IDIOMS)
(* (YEARS OLDER THAN)
   (PLUS)
   IDIOMS)
(* (THESE)
   (THE)
   IDIOMS)
(* (MORE THAN)
   (PLUS)
   IDIOMS)
(* (FIRST TWO NUMBERS)
   (THE FIRST NUMBER AND THE SECOND NUMBER)
   IDIOMS)
(* (THREE NUMBERS)
   (THE FIRST NUMBER AND THE SECOND NUMBER AND THE THIRD NUMBER)
   IDIOMS)
(* (ONE HALF)
   (.5)
   IDIOMS)
(* (PERCENT LESS THAN)
   (PERLESS)
   IDIOMS)
(* (LESS THAN)
   (LESSTHAN)
   IDIOMS)
(* (TWO NUMBERS)
   (THE FIRST NUMBER AND THE SECOND NUMBER)
   SIM)
(* (TWICE)
   (2 TIMES)
   IDIOMS)
(* (TWO NUMBERS)
   SIM)
(* ((* DOLLAR)
    ($ . 1))
   (2 DOLLARS)
   IDIOMS)
(* (CONSECUTIVE TO)
   ((QUOTE 1)
    PLUS)
   IDIOMS)
(* (LARGER THAN)
   (PLUS)
   IDIOMS)
```

```
(* (PER CENT)
   (PERCENT)
   IDIOMS)
(* (HOW MANY)
   (HOWM)
   IDIOMS)
(* (SQUARE OF)
   (SQUARE)
   IDIOMS)
(* (($ . IS)
    MULTIPLIED BY)
   (TIMES)
   IDIOMS)
(* (($ . IS)
    DIVIDED BY)
   (DIVBY)
   IDIOMS)
(* (THE SUM OF)
   (SUM)
   IDIOMS)
(* ($)
   (/ (*S MONID 1))
   *)
```

Het programma STUDENT is aan het werk te zien in Palo Alto bij de Xerox Corporation waar Bobrow werkt. Op het rekencentrum van de universiteit van Utrecht echter is ook de oorspronkelijke versie geïmplementeerd. Aan een implementatie van een Nederlandse STUDENT wordt gewerkt.

- ELIZA

Een tweede, minstens even beroemd vraag/antwoord-systeem is het programma van Weizenbaum: ELIZA. ELIZA werd uitgebreid voorgesteld in hoofdstuk 2, waarnaar bij deze wordt verwezen. Toen echter moest de programmering ervan open blijven, tot de middelen en technieken klaar stonden om ELIZA te programmeren. Net als STUDENT is ELIZA een puur op patroonherkenning gebaseerd geheel. In STUDENT moest echter nog een beetje wiskunde worden bedreven. ELIZA is alleen het opzoeken van sleutelwoorden en dan wat substituties aanbrengen, eventueel wat transformaties (bijvoorbeeld als een van de twee vraagt: ben je...; dan moet de ander terugkomen niet met 'je', maar met 'ik').

- Programmeren van ELIZA in METEOR

Stel dat we ELIZA niet toestaan over haarzelf te praten (ze is immers een niet-directieve rogeriaanse therapeute!), dan moet iedere keer wanneer 'je' of 'u' voorkomt in de input-tekst de reactie komen: We zouden het over jouw probleem hebben, niet over het mijne.
De METEOR regel daarvoor is eenvoudig:

```
(* ( JE )   ( WE ZOUDEN HET OVER JOUW PROBLEEM HEBBEN EN
             NIET OVER HET MIJNE ) *)
(* ( U )    ( WE ZOUDEN HET OVER JOUW PROBLEEM HEBBEN EN
             NIET OVER HET MIJNE ) *)
```

We kunnen ons voornemen dat de cliënt geen vragen mag stellen. Het antwoord daarop zal dan bijvoorbeeld zijn: Het is de bedoeling dat ik de vragen stel, weet je nog?

Deze kwestie kunnen we ophangen aan de matching van een vraagteken of aan de vragende volgorde van werkwoord en persoonsvorm. In het eerste geval lopen we de kans op een misser wanneer de cliënt vergeet een vraagteken toe te voegen, in het tweede geval moet het programma het werkwoord in de zin kunnen vinden. In het voorgaande is uiteengezet wat nodig is om het werkwoord te vinden (zie woordklastoekenning), dus we moeten nu beslissen om een programma voor woordklastoekenning in te voegen of om er een puntje aan te zuigen. We zouden kunnen combineren tussen vraagteken en een serie te verwachten werkwoorden zoals ben je; kan je; mag je; moet je; durf je; zul je; hoop je; denk je.

In dat laatste geval zien de METEOR regels er als volgt uit:

```
(*   ( ? )              QU)
(*   ( BEN JE )         QU)
(*   ( KUN JE )         QU)
(*   ( MAG JE )         QU)
(*   ( MOET JE )        QU)
(*   ( DURF JE )        QU)
(*   ( ZUL JE )         QU)
(*   ( HOOP JE )        QU)
(*   ( DENK JE )        QU)
(QU  ( $ )   ( HET IS DE BEDOELING DAT IK DE VRAGEN STEL
               WEET JE NOG? )  *)
```

- De sleutelwoorden

ELIZA reageert op een serie sleutelwoorden zoals 'voel', 'denk', enz., men kan er bij denken wat men wil. Daarop moet het programma antwoorden in de trant van: waarom voel je.... . Voorbeeldzin: Ik voel me niet best. Reactie: Waardoor voel je je niet best?
De input-zin verandert dus nogal flink:
- ten eerste wordt de volgorde veranderd
- ten tweede wordt de behandelende persoon veranderd
- ten derde wordt een stukje van de zin herhaald, namelijk het stukje na 'me'.

De voorbeeldzin was dus: Ik voel me niet best.
De structuur van deze zin is voor ELIZA: IK $ ME $
Waarop ELIZA moet reageren: WAARDOOR $ JE JE $

Met andere woorden de regel voor deze zin is:

(* (IK $ ME $) (WAARDOOR 2 JE JE 4 ?) *)

VRAAG. Welke andere gevallen zijn nu ook al opgelost?

Bij vergelijking met het ELIZA programma dat in BASIC werd geprogrammeerd valt nu op dat het BASIC programma alle werkwoorden een voor een afvraagt. Het METEOR programma heeft op deze wijze ALLE werkwoorden; ook de werkwoorden die niet in de sleutellijst staan gevangen.

Met de werkwoorden uit de lijst van sleutelwoorden kan men ook anders omgaan. Men kan ook vragen na de zin "Ik voel me niet best": "Waar komt denk je dat gevoel vandaan?". Dit is een specifiekere vraag dan de normale opdracht in ELIZA "please elaborate on that". Dit zou men als volgt kunnen programmeren:

(* (VOEL ME) (WAAR KOMT, DENK JE, DAT GEVOEL
VANDAAN ?) *)

VRAAG. Welke andere woorden uit de sleutellijst lenen zich voor een soortgelijke behandeling als VOEL/GEVOEL?

De kunst is ELIZA door te laten vragen. Daarbij kan men aan allerlei strategieën denken. Een ervan is te onthouden waar het gesprek over ging. Daartoe kan men wat belangrijke sleutelwoorden uit de vragen op een SHELF zetten en die dan als dictionaire aflopen. Komt hetzelfde woord nog een keer voor, dan kan ELIZA antwoorden: "Je hoeft jezelf niet te herhalen, ik heb heus wel naar je geluisterd". De organisatie van dit gedeelte kan worden samengesteld uit de technieken die werden geleerd in de uiteenzetting over METEOR zelf.

Verder kan men, wanneer er niets wordt gematched, een serie standaardopmerkingen laten maken in de trant van:
"Ik luister", "Spreek je maar rustig uit", "Ga maar rustig door", "Ben je daar echt zeker van?", enz.
Dit soort opmerkingen moeten natuurlijk niet al te dikwijls achter elkaar gemaakt worden. Daarom is het goed daarvoor een dobbelsteen te hebben, een zogenaamde r a n d o m g e n e r a t o r. Deze random generator levert een serie willekeurige getallen op, al naar gelang het getal neemt ELIZA een van de standaardopmerkingen.

De random generator moet in LISP worden geschreven en worden ingebouwd in ELIZA. Hij kan worden overgenomen in de vorm van de functie WILLEKEUR die al aan de orde kwam.

3.3.3.4 "He ate at the beehive" of: Tekstproduktie per computer

In het eerste hoofdstuk is uitgebreid aan de orde geweest hoe men standaardbrieven, rapporten en polissen per computer maakt. Dit gebied van automatische tekstproduktie per computer zou men kunnen samenvatten onder het begrip: 'kantoorteksten per computer'.

In Computerworld van augustus 1980 stond echter een andere tekstproducerende computer te kijk. Er was namelijk een uitgever die bekend maakte dat een computer geheten Melopomene van de Jagiellonische Universiteit in Krakow een roman had geschreven. Deze roman getiteld 'Bagabone, Hem 'I Die Now', werd door de uitgeverij Vantage Press Inc. gepubliceerd. De titel is volgens de uitgever een dialectuitspraak van "Bagabone ligt op sterven'. De heer Bruce R. Kent van de afdeling voorlichting vertelde dat Melopomene een fonetisch alfabet had geleerd van 44 klanken. Dat hij die allemaal geleerd had uit de werken van James Joyce, D.H. Lawrence, wat vrouwelijke schrijvers en wat 'angry young men' uit de zestiger jaren. Bij deze club had Melopomene ook schrijven geleerd. Het programma zou dan 27 werkwoordpatronen, 5.000 betekenissen uit het Engels, 200 regels uit de systemic grammar (denk aan Winograd) en 200 regels uit de systemic grammar van het Frans hebben gecombineerd en met dat al de roman 'Bagabone, Hem 'I Die Now' hebben geschreven. De roman speelt in de Stille Zuidzee. Hoe de computer aan de kennis over het leven aldaar kwam is niet duidelijk geworden, misschien van de angry young men.

De roman zou zijn gemaakt aan het Instituut voor exacte wetenschappen en technologie van genoemde universiteit. De universiteit heeft echter een dergelijk instituut niet. Ook beschikt heel Polen niet over een computer die groot genoeg is om een dergelijk karwei te klaren. Verder berust het auteursrecht op de roman bij een Engelsman genaamd G.E. Hughes, niet bij de universiteit van Krakow.

Computerworld vond het verhaal van Vantage Inc. toch wat al te fantastisch en ging op zoek naar specialisten op het gebied van het verhalen schrijven met computers. Ze kwamen aan bij James Richard Meehan, die een proefschrift in handelsuitgave getiteld 'The Metanovel: Writing Stories by Computer' het licht had doen zien in 1980. Geen boek over de tekstmachine is compleet wanneer niet de ontdekkingen en ideeën van Meehan erin aan de orde zijn geweest.

- Detectives per computer

Voor Meehan's TALE SPIN, het programma dat hij maakte om verhalen te schrijven met de computer, aan de orde kan komen moet eerst aandacht worden besteed aan de beroemdste voorloper van Meehan, namelijk Sheldon Klein.

Klein en zijn groep van de universiteit van Wisconsin wilde een systeem maken dat het schrijven van verhalen kon nabootsen. Ze gebruikten daarvoor de theorieën van Vladimir Propp die een serie

Russische volksverhalen en sprookjes had geanalyseerd. Propp heeft ook een notatie ontwikkeld voor de gemeenschappelijke stukken in de structuur van de sprookjes en verhalen. Klein keerde de analyse van Propp om en maakte van de analyse een genererend systeem. Hij gebruikte een nieuwe programmeertaal, namelijk MESSY. Een van zijn belangrijkste oogmerken was schrijfsnelheid. Zijn 'Propp-model' schrijft verhaaltjes met een tempo van 128 woorden per minuut. Wat Klein wil is zo efficiënt mogelijk verhaaltjes volgens een vast patroon laten maken.

Voorbeeld: Een programma uit de detective-schrijver van Sheldon Klein

```
%              HITTING SOMEONE OVER THE HEAD WITH A HEAVY
%              OBJECT FOR BLACKMAILING YOU
%
%
$LOOP K9:  H. PICK(HEAVYOBJ);
$RULE:    MOVE H TO EVIDENCE.
          MOVE H TO WEAPON.
          MOVE FEAR TO MOTIVE.
         V BLACKMAIL K.
          INSERT (K MADAT V) (MADAT VERY).
          INSERT (K AFRAID) (AFRAID OF V).
          INSERT (K DECIDE) (DECIDE KILL V).
         DAY IS SUNDAY.
         TIME IS DAWN.
         K GETUP.
         K GOTO HALL.
         MX QQ = 1
         HALL DARK.
         K HIDENO.
         K HAVE H.
          INSERT (V AWAKEN) (AWAKEN EARLY).
          INSERT (V EARLY) (EARLY USUALLY).
         V GOFOR WALK.
         K WAITFOR V.
         K SURPRISE V.
          INSERT (K HIT V) (HIT WITH H).
          INSERT (V GROAN) (GRAON WEAKLY).
         V DIE;
$RULE:    INSERT (K REMOVE FPRINTS) (FPRINTS ON H);
         (K IQ)/150;
$RULE:   ($ENDGROUP)
         K RETURN TO BEDROOM;
$ENDLOOP;
```

Dit programma levert de volgende tekst op:

```
DR. BARTHOLOMEW HUME BLACKMAILED EDWARD.
EDWARD WAS AFRAID OF DR. HUME.
LORD EDWARD DECIDED TO KILL DR. BARTHOLOMEW HUME.
THE DAY WAS SUNDAY.
THE TIME WAS THE SUNRISE.
LORD EDWARD GOTUP.
LORD EDWARD WENT TO THE DARK CORRIDOR.
LORD EDWARD HID.
EDWARD HAD A CANDLE HOLDER.
DR. BARTHOLOMEW HUME AWAKENED EARLY.
DR. BARTHOLOMEW HUME WAS USUALLY EARLY.
DR. HUME WENT FOR THE WALK.
EDWARD WAITED FOR HUME.
LORD EDWARD SURPRISED HUME.
EDWARD HIT DR. BARTHOLOMEW HUME WITH THE CANDLE HOLDER.
DR. BARTHOLOMEW HUME GROANED WEAKLY.
DR. HUME DIED.
EDWARD RETURNED TO THE BEDROOOM.
```

Als men de tekst van het computerprogramma met de geproduceerde tekst vergelijkt, blijkt al vrij snel dat het computerprogramma niet veel anders doet dan oplepelen wat er letterlijk is ingestopt. Het programma heeft weinig mogelijkheden om zelf iets te concluderen, samen te stellen, af te leiden en om te vormen. Men kan dan ook rustig stellen dat dit computerprogramma weinig uitstaande heeft met het echt zelf maken van verhalen. Het model is evenveel een model van menselijke activiteit als een kopieerapparaat een simulatie geeft van de inhoud van het stuk dat wordt gekopieerd. Het zal het kopieerapparaat een zorg zijn wat de inhoud van de gekopieerde stukken is!

Wil men zo komen tot het nadoen van de computer van schrijfactiviteiten dan is het wel de allerlangste weg; men zal dan eerst volledig de te schrijven roman moeten schrijven en hem dan nog eens door de computer moeten laten afdrukken. Er moet meer mogelijk zijn dan dit magere model wanneer we in ogenschouw nemen wat er allemaal niet door een computer van teksten kon worden begrepen, en dan echt begrepen zodat er een vraag/antwoord-systeem op kon worden gebouwd. De poging om verder te gaan werd ondernomen door Meehan met zijn TALE SPIN.

• TALE SPIN

Wil men verhalen schrijven met de computer, wat moet de computer dan weten en wat moet hij kunnen? Om te beginnen is deze vraag wat te algemeen gesteld. Laten we beginnen met de term 'verhalen', wat we vernauwen tot eenvoudige verhaaltjes zoals de fabels van Aesopus (De vos en de raaf, bijvoorbeeld). In dit soort verhalen gaat het om dieren die menselijk gedrag vertonen. Dit gedrag heeft dan weer te maken met een moraal die in het verhaal gaat zitten. Nu kunnen we al iets specificeren over wat de computer moet weten en over wat hij moet kunnen.

De computer moet een model hebben van de tastbare wereld en hij moet kennis hebben van menselijk gedrag. Hij moet karakters kunnen nabootsen die motieven tot handelen hebben, die emoties hebben en die verbindingen met andere karakters hebben (relaties zo men wil). Juist door de kennis van de motieven, de emoties en de betrekkingen zal de computer de karakters handelend kunnen doen optreden ofwel met een ander woord daardoor zal hij ze kunnen nabootsen/simuleren.

Maar de computer moet meer weten. Hij moet ook weten wat verhalen zijn, wat interessant is en wat samenhangend is. Het programma TALE SPIN is een programma waarmee al deze zaken kunnen worden uitgezocht. Een programma dat zo flexibel is dat het kan worden bijgesteld wanneer er iets misgaat op een van de onderdelen die net als belangrijk aan de orde werden gesteld.

In TALE SPIN gaat het over mensen en pratende dieren. De belangrijkste motivaties komen uit lichamelijke behoeften zoals honger. Bergen, holen, bossen en dergelijke zijn in de wereld van TALE SPIN voorhanden. Deze wereld ligt niet van te voren vast. Dat wil zeggen: het gaat bij TALE SPIN niet zoals gewoonlijk wanneer de computer gebruikt wordt bij de simulatie van de materiële wereld. Daar is het immers normaal zo, dat wanneer een stukje uit de kenniswereld van de computer wordt geactiveerd, alles direct volledig wordt klaargezet. Dat wil zeggen, zou het gaan over bijvoorbeeld Chicago, dan wordt in de normale computersystemen à la Schank of Winograd direct het hele s c r i p t of hoe men het ook noemt klaargezet, bijvoorbeeld het hele gebied 'de topografie van Amerika'. De redenering van Meehan gaat als volgt. Stel dat iemand de afstand tussen Londen en Parijs moet schatten, dan zijn er allerlei soorten technieken om dat te doen. Iemand zal bijvoorbeeld denken: Hoe lang duurt de vlucht van Londen naar Parijs? Dit soort berekeningen is voor de mens veel voor de hand liggender dan op te zoeken welke x- en y-coördinaten Londen en Parijs hebben en via worteltrekken te berekenen wat de afstand is. Deze laatste manier van doen, die dus gebruik maakt van de topografische kennis van Europa, ligt wel zeer in de lijn van de computer, maar niet in de lijn van de mens.

Verhalen schrijven nu is een dermate complexe, typisch menselijke activiteit, dat het volgens Meehan verkeerd zou zijn een strate-

gie te bedenken die op het lijf van de computer is geschreven en niet op het lijf van degene die men probeert na te bootsen, namelijk de mens. Alle specificaties die boven werden gegeven, zoals karakter, motieven, interessant, samenhangend, zijn allemaal typisch woorden die bij mensen horen en niet bij computers. Anders gezegd, wil je menselijk gedrage nabootsen dan moet je niet met wiskunde komen aanzetten, maar met kunstmatige intelligentie. Meehan is geïnteresseerd in de studie van mensen en niet zoals de informatica bijvoorbeeld in de studie van computers. Is men geïnteresseerd in computers dan zal men al snel denken: Al die vage begrippen als 'interessant', 'motieven', 'gedrag', wat moeten we daar nu mee? Is men geïnteresseerd in mensen dan zal men vertrekken vanuit het punt: Mensen doen het (waarschijnlijk) zo, laten we eens proberen of de computer dat niet kan nadoen. Bij dat laatste standpunt krijgt men heel boeiende opties. Zo bijvoorbeeld de redenering: Het is een feit dat de computer snel en efficiënt rekenwerk kan doen. Dat wil echter nog niet zeggen dat mensen het ook op die manier doen. Dat nu weer betekent dat je, wanneer mensen het anders doen, bij een nabootsing van menselijk gedrag ook de computer het anders moet laten doen. Een voorbeeld was al het schatten van de afstand tussen Londen en Parijs.

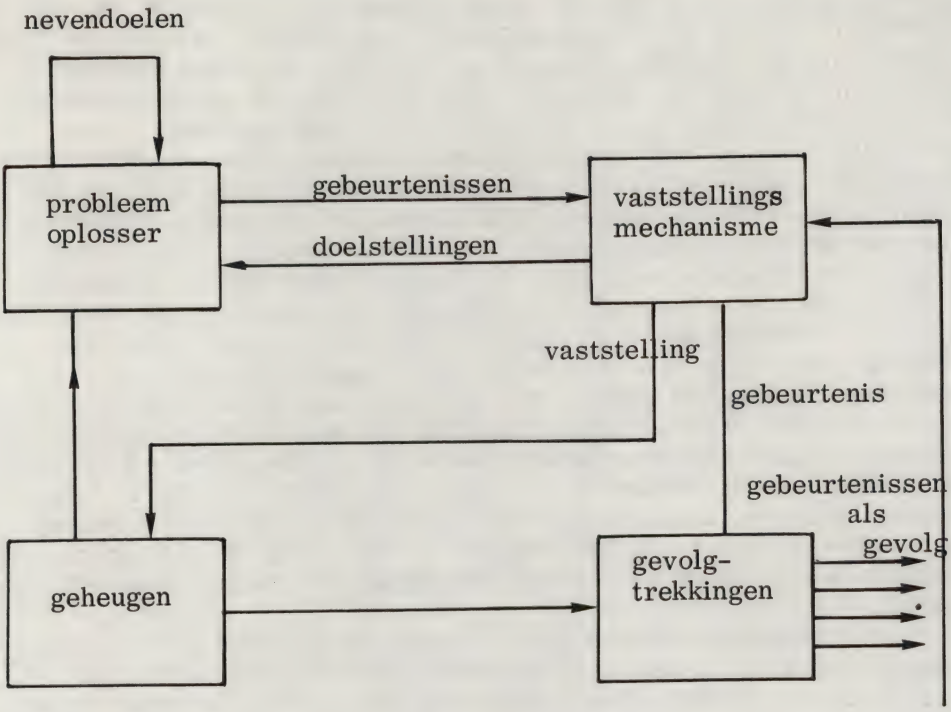
Een kernmoeilijkheid blijft nu dus toch, zoals altijd in dit gebied, hoeveel kennis er in de computer gestopt moet worden. Zolang de doelstellingen zo vaag zijn als 'nabootsen van menselijk gedrag' is er geen grenslimiet te vinden bij het aanbrengen van kennis.

De doelstelling van TALE SPIN was het schrijven van verhalen. Het criterium om nieuwe informatie toe te voegen of informatie aan te brengen is dat op grond daarvan nieuwe verhalen geschreven moeten kunnen worden en verder niets.

Het programma TALE SPIN bestaat uit vier functionele onderdelen. Een probleemoplosser, een mechanisme dat de dingen vaststelt, een geheugendeel en een mechanisme om gevolgtrekkingen te maken. Het verband tussen de vier functionele onderdelen is op p. 221 schetsmatig weergegeven.

- De functionele onderdelen

Op zich is het blokschema dat de structuur van TALE SPIN aangeeft duidelijk. Immers er zullen gebeurtenissen in een verhaal plaatsvinden die moeten worden vastgesteld. Die weer zullen moeten worden getoetst aan de doelstellingen die de personen in het verhaal hebben. Wordt iets vastgesteld, dan moet het ook worden opgeslagen in het geheugen anders heeft het geen zin. Een vaststelling impliceert wellicht ook een nieuwe gebeurtenis. Een nieuwe gebeurtenis zal gevolgen kunnen hebben die hun neerslag vinden in weer nieuwe gebeurtenissen, maar om dat uit te vinden moet eerst de gevolgtrekking worden gemaakt dat inderdaad die nieuwe gebeurtenissen hebben plaats-



gevonden. Tot zover zal alles wel duidelijk zijn, omdat dit ook op abstracte basis wel te volgen is. Een kernpunt om tot een concreet begrip van TALE SPIN te komen ligt in de taak van de probleem oplosser. Men denkt bij de term probleem oplosser namelijk nogal snel aan het gebied problem solving uit de kunstmatige intelligentie. Een gebied waarin problemen uit de logica aan de orde komen.

De belangrijkste programma's om problemen op te lossen zijn gemaakt door Newell en Simon (GPS General Problem Solver), door Fikes, Hart, Nilsson en Sacerdoti (STRIPS) en door Fahlmann (BUILD). Deze klassieke probleemoplossers zijn gebaseerd op de zogenaamde techniek van 'means end analysis'. De truc in deze techniek is dat steeds het verschil wordt bekeken tussen de gewenste staat van de wereld en de nu aanwezige stand van zaken. Dit verschil wordt dan steeds verkleind door een serie veranderingen (transformaties) waarvan bekend is dat ze nuttig zijn. De wereld waar het om gaat is meestal de wereld van een robot met wat blokken. Die robot kan dan de wereld veranderen door zichzelf te verplaatsen of de blokken te verschuiven. Het programma STRIPS zoekt naar waarheden en is dus gebaseerd op predikaat calculus. Daarin krijgt men dan notatie als ATR(Z), hetgeen betekent: de robot (R) is op (AT) plaats Z, of de notatie AT(X Y) wat betekent: object X bevindt zich op plaats Y.

Met predikaat calculus kan men dan opgeven dat een object niet op twee plaatsen tegelijk kan zijn en van daaruit kan men dan weer tot de conclusie komen of X zich op C of op Y bevindt. Andere klassieke probleemoplossers gebruiken soortgelijke technieken. Deze worden dan weer aangewend bij vraag/antwoord-systemen zoals die in een van de vorige paragrafen zijn behandeld.

Wat onder al deze systemen als vooronderstelling zit is de gedachte dat je alles moet terugvertalen naar een serie uiterst eenvoudige zogenaamde primitieven. Met deze primitieven zou men alles (moeten) kunnen representeren.

Deze redenering is aantrekkelijk omdat hij zo eenvoudig is, maar niet omdat hij zo juist is. Het is juist tegen het specifiek menselijke vermogen kennis zo te organiseren dat er nuances en verschil mogelijk is. Men kan dan ook gemakkelijk een serie zinnen of uitspraken vinden die er oppervlakkig gezien hetzelfde uitzien. Neem bijvoorbeeld de volgende zes zinnen.

1. Piet is de bezitter van het huis.
2. Utrecht is de hoofdstad van Utrecht.
3. Vijf is de wortel van vijfentwintig.
4. D is de dominant van A.
5. Vrouwen is het meervoud van vrouw.
6. Deuterium is de tweede mogelijke isotoop van waterstof.

Laten we aannemen dat het hierbij alleen maar gaat om simpele feiten en dat het niet gaat om beeldspraak. Wat we dan zien is dat al deze uitspraken uit verschillende kennisgebieden komen.

Deze kennisgebieden hebben alle hun eigen specifieke kennisorganisatie in het geheugen van mensen. Mensen kijken dus op een verschillende manier tegen die verschillende zinnen aan. Wil je nu op een menselijke manier met deze kennis omgaan, dan moet je bij voorkeur niet al deze feiten op dezelfde manier in een identieke primitieve operator duwen. Natuurlijk kan men dat wel doen, maar dan wordt tegelijk alles op een hoop gegooid, terwijl het voor mensen helemaal niet soortgelijk is. Daarmee ben je dan ook gedwongen alles op dezelfde manier te behandelen. Dat zou dus willen zeggen dat om de hoofdstad van Utrecht te vinden je langs de weg zou moeten gaan waarmee je ook de wortel uit 25 kunt opsporen. Waar zou de mens te vinden zijn die dat zo doet?

Ofwel: het vinden van de dominant van A is heus wat anders dan het vinden van de eigenaar van een huis. Dat zit anders in je geheugen en je doet er ook andere dingen voor om het uit te vinden als het niet in je geheugen zit. Dit laatste nu is probleem oplossen in de termen van Meehan.

De term probleemoplosser op zich verwijst dus op de achtergrond toch weer naar de wiskunde waarvan nu net gesuggereerd was dat die niet tot het meest natuurlijke menselijke gedrag leidt. Ten aanzien van TALE SPIN is dit echter een misverstand. De problemen die de probleemoplosser oplost zijn nu juist problemen van het soort dat we

hebben aangeduid bij het probleem te schatten wat de afstand tussen Londen en Parijs is. De problemen die aan de orde komen zijn alledaagse problemen, geen wiskundige of vaktechnische problemen. TALE SPIN is in feite een serie procedures die leiden tot een bepaald doel, bijvoorbeeld in het geval van de vos en de raaf het doel van de vos het stuk kaas uit de bek van de raaf te krijgen. Hij zal daarbij weinig hebben aan predikaat calculus van welke orde dan ook of aan een Montague grammatica van het Frans of Engels of aan de linguistische universalia waar Chomsky zo verliefd op was. Een probleem kan bijvoorbeeld zijn hoe je van de ene plaats naar de andere komt. De probleemoplosser produceert dan ten eerste de oplossing en zorgt dat de persoon nu weet dat hij op die nieuwe plaats is en niet op een andere, en als nevenprodukt levert de probleemoplosser ook een (als men wil) reisverslag op van de reis zelf, van de voorbereidingen voor de reis inclusief alle contacten en handelingen met andere mensen die nodig waren om tot die reis te komen (bijvoorbeeld een loketbeambte of een chauffeur). Dit nu doet duidelijk denken aan het werk van Schank. Vooral aan het werk dat Schank samen met Abelson heeft gedaan over het maken van plannen en de structuur daarvan. De probleemoplosser van Meehan is dan ook gebaseerd op dit werk van Schank en Abelson. Van Schank is daarin met name de conceptual dependency theorie opgenomen. Men ziet nogmaals hoe baanbrekend dit werk van Schank is geweest.

Nu zou men in eerste instantie kunnen denken dat de beer los is, want je kunt nu van te voren niet alles meer bedenken, dus je kunt ook niet voorspellen wat het systeem gaat doen en dan kun je dus ook nooit echt een probleem oplossen. Toch is dat niet helemaal juist. Het model van Meehan bergt ook een paar vereenvoudigingen in zich.

Eén ervan hebben we al gezien in het voorbeeld dat je om de hoofdstad van Utrecht te vinden niet langs de weg hoeft te gaan waarlangs je de wortel uit 25 vindt. De tweede is dat mensen juist niet alles van te voren hoeven te weten wanneer ze verhalen vertellen. Er zijn dingen die je moet weten wanneer je iets wilt gaan ondernemen. Meehan noemt dat 'plantime preconditions'. Maar er zijn ook dingen die je pas beslist wanneer je bezig bent. Meehan noemt dat 'runtime preconditions'. En hij zegt er dan bij:
"We are not modelling robots on Mars, we can afford to make mistakes to tolerate some problems along the way, because coping with them – not planning but replanning – is also an interesting human characteristic that we want to put in our model."

Dus nogmaals: de probleemoplosser van Meehan sluit aan bij de opvatting van Schank en Abelson dat er meer manieren zijn om problemen op te lossen en dat mensen ook verschillende technieken gebruiken. Dit in tegenstelling tot de klassieke problem solvers die ervan uitgaan dat alles op één manier moet worden opgelost.

In de probleemoplosser van Meehan zitten de volgende onderdelen aangeduid met de namen die Meehan gebruikt:
- DELTA - ACTS
- PACKAGES
- SIGMA - STATES
- ACTION MODULES
- REACTION MODULES
- RELATIONSHIP STATES
- PERSONALITY STATES

Vanaf de term action modules spreken de termen voor zich, aangezien het Engels al duidelijk aangeeft wat de procedures te doen hebben die met die term belast zijn. Anders zit het met de DELTA ACTS, de PACKAGES en de SIGMA STATES. Daarom nog een enkel woord ter verduidelijking van wat die inhouden.

• DELTA ACTS

Hierin wordt kennis georganiseerd die op een speciaal doel gericht is.
Voorbeeld: DELTA PROX
Dit is de procedure die persoon X gebruikt om Y in de buurt van Z te krijgen. Hierin zit dan een serie methoden in volgorde van gecompliceerdheid. Deze noemt Meehan Planboxes.
Planbox O voor DELTA PROX is de vraag:
Denkt X dat Y al in de buurt van Z is?
Dan komt Planbox 1 met de volgende tekst:

Planbox 1 X tries to move Y to Z

preconditions:	X is self movable
	If X is different from Y
	then DPROX (X, X, Y) and DO GRASP (X, Y)
	DKNOW (X where is Z ?)
	DKNOW (X where is X ?)
	DLINK (X loc (Z))
act:	DO PTRANS (X, Y, loc (Z))
postcondition:	Is Y really at Z? (DKNOW could have goofed)
postact:	If X is different from Y then do
	DO NEG GRASP (X, Y)

In deze tekst is goed te volgen dat eerst wordt gevraagd of X niet hetzelfde is als Y, dat er dan wordt gevraagd wat er allemaal te weten is, dan wordt X getransporteerd, waarna nogmaals wordt gecontroleerd of X werkelijk wel op de gewenste plaats staat. De term NEG betekent: doe het niet.

Op vergelijkbare wijze als geschilderd voor het verplaatsen van iemand of iets heeft Meehan DELTA ACTs ontwikkeld voor vertellen (DELTA TELL), voor weten (DELTA KNOW), controleren (DELTA CONTROL) en overtuigen (DELTA PERSUADE).

• SIGMA STATES

SIGMA STATES zijn de definities van lichamelijke behoeften die van tijd tot tijd bevredigd moeten worden. De volgende SIGMA STATES bestaan in TALE SPIN: SIGMA-HUNGER, SIGMA-THIRST, SIGMA-REST, SIGMA-SEX. Waar deze zaken voor dienen zal duidelijk zijn.

• PACKAGES

Een van de packages hebben we behandeld onder de DELTA-ACT, namelijk het PERSUADE PACKAGE. Packages zijn dus gebundelde DELTA-ACTS.

• PERSONALITY STATES

Het belang van de 'personality states' voor het schrijven van verhalen van het soort dat Meehan wilde, kan men inzien, wanneer men het volgende bedenkt. Personality states zijn zoiets als karaktereigenschappen in de trant van 'aardig', 'ijdel', 'intelligent', 'eerlijk'. Stel nu dat iemand aan een ander een verzoek doet, dan zijn er een serie voorwaarden die worden bekeken, vóór de beslissing wel of niet op ingaan valt. Zo zal iemand wellicht eerst bedenken: "Hoe is onze relatie?". Als hij/zij daarop niet kan beslissen, zal hij/zij bedenken welke gevolgen een al dan niet ingaan op het verzoek heeft en tenslotte zal hij/zij beslissen op de overweging: "Ik wil aardig zijn" (of iets dergelijks). Daarom is het van belang te weten of iemand aardig is of wil zijn.

Een ander voorbeeld zou kunnen zijn het feit dat een compliment veel meer indruk maakt op iemand die ijdel is dan op iemand die dat niet is. Als iemand in een valstrik trapt, kan hij worden uitgelachen, zichzelf stom vinden (of iets dergelijks) en daarop zal hij/zij dan al naar gelang zijn karakter heel fel reageren, met humor of gewoon onverschillig.

• De gevolgen

Veel in wat verteld wordt, is niet expliciet wat de gevolgen betreft. We zagen dat al bij de beschrijving van SAM. Dat is de reden dat er ook een gevolgtrekker in TALE SPIN moet zitten. We volstaan met voorbeelden om te laten zien welke soorten gevolgtrekkingen gemaakt kunnen worden door TALE SPIN.

1 Specificaties (de gevolgtrekkingen in hoofdletters)
Piet pakt een steen
Piet slaat Jan
PIET SLAAT JAN MET EEN STEEN

2 Oorzaak
Piet slaat Jan met een steen
PIET IS WAARSCHIJNLIJK KWAAD OP JAN

3 Resultaat
Mien gaf jan een postzegel
JAN HEEFT EEN POSTZEGEL

4 Motivatie
Piet slaat Jan
PIET ZAL JAN WAARSCHIJNLIJK PIJN WILLEN DOEN

5 Mogelijk maken
Beatrix ging naar Parijs
WAAR HAD ZE HET GELD VANDAAN?

6 Functies
Piet wil het boek hebben
PIET ZAL HET WEL WILLEN LEZEN

7 Voorspellen van mogelijk maken
Jan keek in zijn kookboek om te zien hoe hij een appeltaart moest bakken
JAN ZAL NU BEGINNEN EEN APPELTAART TE BAKKEN

8 Mogelijkheid ontnomen
Mien kon de finish van de wielrenners niet zien
Ze stootte de man die voor haar stond aan
DE MAN BELEMMERDE HAAR UITZICHT

9 Tussenkomst
De baby kroop de straat op
Mien rende er achteraan
MIEN WILDE EEN ONGELUK MET DE BABY VOORKOMEN

10 Actie voorspellen
Mien wilde wat spijkers hebben
ZE GING NAAR DE HOBBYSHOP

11 Doorgeven van kennis
Piet vertelde Jan dat Reginald dol was op Beatrix
JAN WEET DAT REGINALD DOL IS OP BEATRIX

12 Plichten
Piet zag Beatrix op dinsdagmorgen op het strand
WAAROM ZAT ZE NIET THUIS TE STUDEREN

13 Voortduren van iets
Jan gaf Mien gisteren een boek in handen
Heeft ze het nu nog vast?
WAARSCHIJNLIJK NIET

14 Kenmerk
Ronald's luier is nat
RONALD ZAL WEL EEN BABY ZIJN

15 Situaties
Beatrix gaat naar een gemaskerd bal
ZE ZAL WEL VERKLEED GAAN

16 Bedoeling
Mien kon niet over het hek springen
WAAROM WILDE ZE DAT?

Voor TALE SPIN worden alleen de gevolgtrekkingen gebruikt die te maken hebben met de gevolgen die handelingen hebben voor het verdere verloop van het verhaal. Andere gevolgtrekkingen die voor een programma als SAM wel nodig zijn blijven hier achterwege. Althans met TALE SPIN moet zover geëxperimenteerd worden dat alleen de gevolgtrekkingen overblijven die iets met handelingen en het verhaal te maken hebben. Meehan heeft er ook mee geëxperimenteerd, zoals zal blijken uit de voorbeelden.

De toespitsing op het verhaal heeft ook tot gevolg dat alle oude toestanden verdwijnen, zodra er een nieuwe is. Stel bijvoorbeeld dat Jan en Piet in Amsterdam zijn; dat Jan Piet uitzwaait op het Centraal Station en dat hij hem een half uur later weer nodig heeft. Dan is het niet erg voor de hand liggend dat hij hem in Amsterdam gaat zoeken. De informatie over hun verblijf in Amsterdam moet verdwenen zijn op het moment dat Jan Piet gaat zoeken, anders gaat hij toch eerst in Amsterdam zoeken. In een van de voorbeeldverhalen zal blijken waar het toe leidt wanneer de toestand van de informatie niet wordt bijgehouden.

- Het bepalen van het niveau van vertellen

Behalve alle details die in het bovenstaande aan de orde komen is er nog een belangrijk aspect dat met het vertellen van verhalen te maken heeft. Dit aspect is van belang voor de mate waarin het verhaal interessant en samenhangend is.

Laten we ons verplaatsen naar een schrijver die een verhaal wil gaan schrijven dat heet "De beklimming van de Matterhorn". Het probleem van dit verhaal is duidelijk: namelijk hoe kom je bovenop die berg? Maar van welke gezichtshoek uit zal die schrijver het gaan beschrijven? Als gids voor de aankomende alpinist? Als een avontuur in de open lucht? Als een verhaal over de mens die de natuur bedwingt? Als een verhaal over het eeuwigdurende conflict tussen mens en natuur? Of als een verhaal over een conflict dat alleen maar wordt ingevuld met een concrete beschrijving van de Matterhorn als een soort metafoor voor iets dreigends, etc. ?

Deze uitgangspunten noemt Meehan het niveau van vertellen.

Het is duidelijk dat boven aangeduide uitgangspunten hun geldigheid hebben. Wil men echter een roman schrijven met de computer dan moeten er wel beslissingen genomen worden. Het is dus de kunst, in de wirwar van mogelijkheden een nuttig uitgangspunt van vertelniveau

te vinden. Meehan stelt de volgende test voor om een dergelijk nuttig uitgangspunt te vinden. Stel je de schrijver voor die op een goede dag zegt: "Ik ga het verhaal schrijven: We beklommen de Mount Everest." Tegen zichzelf zegt hij dan: "Vandaag ga ik schrijven over ----". Dan zal hij toch hoogstwaarschijnlijk niet invullen 'Conflict'. Maar hij zal evenmin zeggen "hoe je klimt langs de wanden van de Mount Everest". Is het één veel te algemeen, het ander is veel te specifiek. Het is duidelijk dat het te kiezen niveau ergens tussen die twee in ligt. Maar hoe kom je nu aan dat precieze niveau?

Een ander voorbeeld om dat op te sporen is: "Hoe kunnen we MacDonald op de hoek beschrijven in opklimmende orde van abstractie?". Als een vreet-tentje, een soort restaurant, een zelfbedieningszaak, een gebouw dat mensen gebruiken, een fysiek object. Wat is nu de meest nuttige kenschetsing van de MacDonalds? Dat hangt af van degene die de beschrijving moet maken. Voor een klant is het van belang dat je er kunt eten. Voor een binnenhuisarchitect dat het een gebouw voor mensen is. Een metselaar zal het een zorg zijn of mensen het gebouw gebruiken. En in de woorden van Meehan: "And maybe Martians and computational linguists care that they are physical objects".

Hieruit kan de conclusie worden getrokken dat de meest nuttige beschrijving, het meest nuttige vertrekpunt van de vertelling niet altijd op hetzelfde niveau hoeft te liggen en dat gedurende de vertelling ook het niveau of zo men wil het perspectief van waaruit verteld wordt, moet kunnen veranderen. Dit nu gebeurt in werkelijkheid bij het schrijven ook.

Willen we de Matterhorn niet gebruiken voor een dagje uit in de natuur, maar voor een verhaal in de geest van 'mens tegen natuur', dan zal de berg dienen te verschijnen als kwaadwillig tegenover de klimmers of we zouden een van de klimmers een soort vermenselijking van de boze berg kunnen maken. Het eind van de tocht zal, als het goed afloopt, een persoonlijke triomf voor de (overige) beklimmers zijn, etc.

- Twee manieren van verhalen vertellen met TALE SPIN

Op twee manieren kan men met TALE SPIN omgaan. Men kan hem laten lopen en dan zien wat er gebeurt. Dat wil zeggen TALE SPIN kan de lezer vragen stellen en als de lezer die vragen beantwoordt, vult TALE SPIN zelf de geschiedenis in. Dit is de aanpak van onderaf. Het enige wat bekend is voor TALE SPIN is rationeel gedrag in de termen die boven werden behandeld. Deze manier van werken met TALE SPIN wordt de bottom-up mode genoemd.

De andere manier is dat TALE SPIN van de andere kant af werkt. Dan begint TALE SPIN met een boodschap of een soort moraal, kiest zelf het gebied waarin het moet plaatsvinden en richt zelf de wereld in zoals dat bij de doelstelling past. Hier is dus TALE SPIN de alwe-

tende schrijver. Deze methode van werken met TALE SPIN is dus bekeken van bovenaf. Hij wordt dan ook genoemd: top-down mode.

- Verhalen van TALE SPIN

In het onderstaande worden verhalen afgedrukt, sommige in de vertaling die met TALE SPIN is gemaakt. De eerste drie verhalen zijn correcte verhalen uit de definitieve fase van ontwikkeling, de tweede serie van vier verhalen zijn verhalen uit de ontwikkelingsfase van TALE SPIN, waarbij belangrijke zaken nog moesten worden geïmplementeerd, ja zelfs ontdekt.

- Voorbeeld 1 in top-down mode

De vos en de raaf

Er was eens een oneerlijke vos met de naam Henry die in een hol woonde. En er was een ijdele kraai die Joe heette en in een iep woonde. Joe had een stuk kaas in zijn bek en zat er mee op een tak. Op een dag kwam Henry uit zijn hol en liep door de wei naar de iep. Hij zag Joe de kraai met de kaas en hij kreeg honger. Hij dacht dat hij de kaas kon krijgen als Joe ging praten, dus zei hij tegen Joe dat hij zo van zingen hield en dat hij hem graag zou horen zingen. Joe voelde zich zeer gevleid door Henry en begon te zingen. De kaas viel uit zijn bek op de grond. Henry pakte de kaas en zei dat Joe een stommeling was. Joe was kwaad en vertrouwde Henry nooit meer. Henry liep terug naar zijn hol.

- Voorbeeld 2, een stukje in bottom up mode

```
PLEASE CHOOSE ONE OF THE FOLLOWING
1 VERBOSE MODE
2 NOT SO VERBOSE MODE
2 PREPACKAGED PLOT

*1
(gekozen is dus de verbose mode)

WELCOME TO TALE SPIN
CHOOSE ANY OF THE FOLLOWING CHARACTERS FOR THE STORY
(BEAR BEE BOY GIRL COW CROW ANT HEN LION DOG WOLF MOUSE
 CAT GOAT CANARY)

*(BEAR BEE BOY CANARY)
(gekozen worden dus vier handelende personen)

CREATING JOHN BEAR

CREATING A NEW CAVE (*CAVE 0)
```

```
CREATING A NEW MOUNTAIN (*MOUNTAIN*0)

JOHN BEAR IS AT A CAVE

CREATING IRVING BEE
CREATING SOME NEW HONEY (*HONEY*0)
CREATING A NEW BEEHIVE (*BEEHIVE*0)
CREATING A NEW APPLE TREE (*APPLE TREE*0)
CREATING A NEW GROUND (*GROUND*0)
CREATING A NEW VALLEY (*VALLEY*0)

A BEEHIVE IS AT AN APPLE TREE
IRVING BEE IS AT THE BEEHIVE
SOME HONEY IS AT THE BEEHIVE
IRVING BEE HAS THE HONEY

CREATING SAM ADAMS
CREATING A NEW HOUSE (*HOUSE*0)
CREATING A FRONT DOOR (*DOOR*0)
CREATING A NEW VALLEY (*VALLEY*0)

SAM ADAMS IS AT A HOUSE

CREATING A NEW NEST (*NEST*0)
CREATING A NEW REDWOOD TREE (*REDWOODTREE*0)
CREATING A NEW GROUND (*GROUND*0)
CREATING A NEW MEADOW (*MEADOW*0)
CREATING WILMA CANARY

WILMA CANARY IS AT A NEST

CHOOSE ANY OF THE FOLLOWING PROPS
(BREADCRUMS CHEESE BASEBALL)

*NIL
(er wordt dus niets gekozen nu)

CHOOSE ANY OF THE FOLLOWING MISCELLANEOUS ITEMS
(BERRIES FLOWER RIVER WORM)

*(BERRIES WORM)

CREATING SOME NEW BLUEBERRIES (*BLUEBERRIES*)
CREATING A NEW BUSH (*BUSH*0)
```

```
CREATING A NEW MEADOW (*MEADOW*1)
CREATING A NEW PLACE IN MEADOW (*MEADOWPOINT*1)

              SOME BLUEBERRIES ARE AT A BUSH

CREATING A NEW WORM (*WORM*0)
CREATING A NEW PATCH OF GROUND (*PATCH*0)
CREATING A NEW MEADOW (*MEADOW*2)
CREATING A NEW PLACE IN MEADOW (*MEADOWPOINT*2)

              A WORM IS AT A PATCH OF GROUND

WHO KNOWS ABOUT THE BLUEBERRIES?
1: WILMA CANARY          2: SAM ADAMS          3: IRVING BEE
4: JOHN BEAR

*4                JOHN BEAR KNOWS THAT THE BLUEBERRIES ARE AT
                  THE BUSH

HOW HUNGRY IS JOHN BEAR?
1: VERY   2: SOMEWHAT   3: NOT VERY   4: NOT AT ALL

*4                JOHN BEAR IS NOT AT ALL HUNGRY
                  Anders gaat hij gelijk eten!

WHO KNOWS ABOUT THE WORM?
1: WILMA CANARY     2: SAM ADAMS     3: IRVING BEE
4: JOHN BEAR

*2                SAM ADAMS KNOWS THAT THE WORM IS AT THE PATCH
                  OF GROUND
                  JOHN BEAR IS SOMEWHAT HUNGRY
                  JOHN BEAR WANTS TO GET SOME BERRIES
                  JOHN BEAR WANTS TO GET NEAR THE BLUEBERRIES

IN CREATING A VALLEY, WE CAN MAKE UP A NEW ONE OR USE AN OLD
ONE. DO YOU WANT TO USE ANY OF THESE?
1: *VALLEY*0      2: *VALLEY*1
--  DECIDE:       *YES
PLEASE TYPE AN INTEGER BETWEEN 1 AND 2
*1

CREATING A VALLEY-PASS BORDER (*BORDER*6)
CREATING A NEW VALLEY-MEADOW BORDER (*BORDER*7)
CREATING A NEW PASS-VALLEY BORDER (*GATE*0)
CREATING A NEW PASS (*PASS*0)
CREATING A NEW MEADOW-VALLEY BORDER (*BORDER*8)
CREATING A NEW MEADOW-VALLEY BORDER (*BORDER*8)
CREATING A NEW CAVE EXIT (*DOOR*2)
```

CREATING A NEW CAVE ENTRANCE (*DOOR*3)

JOHN BEAR WALKS FROM A CAVE ENTRANCE TO THE
BUSH BY GOING THROUGH A PASS THROUGH A VALLEY
THROUGH A MEADOW.

JOHN BEAR TAKES THE BLUEBERRIES.
JOHN BEAR EATS THE BLUEBERRIES.
THE BLUEBERRIES ARE GONE.
JOHN BEAR IS NOT VERY HUNGRY.
THE END.

- Voorbeeld 3: Top-down mode

Joe de beer en Jack de beer

Er waren eens twee beren, Jack en Joe genaamd, en een bij die Sam heette. Jack kon heel goed opschieten met Sam, maar Joe die een oneerlijke beer was, had hij altijd ruzie. Op een dag had Jack honger. Hij wist dat Sam de bij honing had en dat hij hem misschien zover kon brengen dat hij hem wat honing gaf. Hij kwam uit zijn hol, liep de berg af, door het dal, over de brug naar de eik waar Sam de bij woonde. Hij vroeg Sam om wat honing. Sam gaf hem wat. Toen kwam Joe de beer naar de eik gelopen en zag dat Jack wat honing in zijn poten had. Hij dacht dat hij misschien wat honing kon bemachtigen als Jack hem op de grond zette, dus zei hij dat hij dacht dat Jack niet erg hard kon lopen. Jack nam die uitdaging aan en besliste dat hij even hard zou gaan lopen. Hij zette de honing neer en rende de brug over het dal in. Joe greep de honing en wandelde naar huis.

- Verhalen, 2: Toen het nog niet goed ging

Ook hier gaat het weer om verhalen die TALE SPIN echt gemaakt heeft, zij het om de vertaling ervan.

- Voorbeeld 1

Op een dag had Joe de beer honger. Hij vroeg zijn vriend Irving de vogel waar honing was. Irving vertelde hem dat er een bijenkorf in de eik zat. Joe werd kwaad en dreigde Irving een klap te geven als hij niet zei waar honing was.

VRAAG. Wat ontbrak aan TALE SPIN zodat dit verhaaltje eruit kon komen?

- Voorbeeld 2

Er was eens een oneerlijke vos en een ijdele kraai. Op een dag zat de kraai in zijn boom met een stuk kaas in zijn bek. Hij merkte op dat hij het stuk kaas vast had. Hij kreeg honger en slokte de kaas op. De vos liep naar de kraai. Einde.

VRAAG. Wat ging er mis in dit verhaal en wat moest er dus nog aan TALE SPIN gebeuren om de echte 'de vos en de raaf' te krijgen?

• Voorbeeld 3
Joe de beer had honger. Hij vroeg Irving de vogel waar honing was. Irving wilde hem dat niet vertellen, dus bood Joe hem aan dat hij hem een worm zou brengen, als hij hem wilde zeggen waar honing was. Irving vond het goed. Maar Joe wist niet waar hij wormen kon vinden, dus vroeg hij het aan Irving, die dat niet wilde zeggen. Daarom bood Joe hem aan een worm voor hem te halen als hij zou zeggen waar hij een worm kon vinden. Irving vond het goed. Maar Joe wist niet waar hij wormen kon vinden, dus vroeg hij het aan Irving, die het niet wilde zeggen. Dus bood Joe aan Irving aan dat hij hem een worm zou brengen als hij hem zou vertellen waar een worm te vinden was ..... .

VRAAG. Wat ging er fout en wat moest er nog aan TALE SPIN worden gedaan om dit te voorkomen in de toekomst?

• Voorbeeld 4
Op een dag had Joe de beer honger. Hij vroeg zijn vriend Irving de vogel waar honing te vinden was. Irving zei hem dat er een bijenkorf in de eik zat. Joe wandelde naar de eik. Hij at de bijenkorf op.

VRAAG. Wat ging er mis en hoe moest het hersteld worden in TALE SPIN?

3.3.3.5 Mice run under the snow: Gedichten per computer

Een van de eerste toepassingen op het gebied van automatische teksten schrijven is de produktie van gedichten. Daarbij zijn drie modellen tot nu toe voorgesteld. Het eerste model is het dobbelsteen-model; het tweede het zinvariatie-model, het derde het filter-model.

1. Dobbelen: de geblinddoekte muze

In de vroege twintiger jaren ontstond in Parijs, Berlijn en Genève een stroming in de kunst die Dadaïsme genoemd wordt. In deze stroming werd driftig geëxperimenteerd met allerlei vormen van basismateriaal voor de kunsten zelf. In de poëzie leidde dat bijvoorbeeld tot het volgende experiment. Men deed een blinddoek voor, nam een krant en een rood potlood. Streepte met het rode potlood in de krant geblinddoekt stukken aan. Dan werd de blinddoek afgenomen. De rood onderstreepte woorden en zinnen werden onder elkaar geplaatst, waarop een nieuw gedicht was ontstaan. Uit deze vorm van experimenteren ontstond later de zogenaamde toevalskunst, met een vreemd woord aleatorische kunst, die met name in de muziek van belang werd. De keuze binnen het proces van gedichten-produceren werd op deze wijze overgelaten aan willekeur. Men werkte overigens met basismateriaal dat uit complete teksten bestond (kranteteksten bijvoorbeeld).

Dit toevalsmodel was het eerste dat werd nagebootst, niet zozeer uit het bewustzijn dat men nu Dadaïstische poëzie ging produceren, maar veeleer uit de opvatting: gedichten dat is toch maar willekeur en dat kunnen wij met de computer ook. Het waren dan ook steevast wiskundigen die dit soort teksten maakten. Ingrediënten waren: een woordenboek en een dobbelsteen, ofwel een random generator. Met de random generator zocht de computer gewoon in het woordenboek naar woorden en plaatste ze onder of naast elkaar.

Deze vorm van gedichten genereren kan met de kennis die in dit boek werd aangeboden direct in een concreet programma in LISP worden omgezet.

Een variant hierop vormen de activiteiten van Bonnie Nash-Webber (1974), van het beroemde software house Bolt Beranek and Newman te Cambridge (Massachusetts). Zij verkoopt door de computer gegenereerde Mantra's voor de Transcendente Meditatie. Overigens is dit zo gemakkelijk met de zinnengenerator uit dit boek dat het onderwerp in kringen van Kunstmatige Intelligentie, waar steeds LISP gebruikt wordt, lachwekkend is. De auteur voert in desbetreffende publikatie op dat zij dit werk doet met Wry Writer (= Wrange Schrijver) en zelf veranderde zij haar voornaam in Gnash, enfin ..... .

OPGAVE. Schrijf een programma in LISP dat met het dobbelsteenmodel gedichten maakt.

2. Het zinvariatie-model

De grote zwakte van het dobbelsteenmodel is het onderliggende idee als zou een dichter voor een woordenboek gaan zitten en dan maar wat prikken. Dat er geen verbanden hoeven te bestaan tussen die woorden en dat een dichter niet vanuit betekenissen en betekenisstructuren zou werken is in tegenspraak met iedere taalpsychologische realiteit. Mensen handelen alleen maar wanneer dat handelen betekenis voor ze heeft en wanneer in die handeling betekenis geproduceerd kan worden. Gedichten die niets betekenen bestaan niet. Dat is ook de valstrik van de dobbelsteen-gedichten: zodra iemand ze leest, gaat hij/zij betekenis aan die gedichten toekennen.

Ook het feit dat een dichter alleen met woorden zou werken is een grove versimpeling die voortkomt uit het feit dat wiskundigen taal zien als een verzameling (van woorden), wat taal beslist niet is. Toevallig hebben wij in onze g e s c h r e v e n taal eenheden (zogenaamde 'woorden') met elkaar afgesproken die we door spaties omgeven. In andere geschreven talen zit dat echter anders. Er zijn talen waar alleen hele zinnen worden opgeschreven. Daar ervaart men kennelijk dat wat bij ons hele zinnen zijn als de basiseenheden, ofwel als 'woorden'.

Het is duidelijk dat gedichten niet alleen uit woorden bestaan, maar ook uit aan strikte voorschriften gebonden versregels, uit zin-

nen, uit strofen, uit gedachten, uit beelden, enz. En ofschoon de meerderheid van de computerpoëzie in de dobbelsteen is blijven steken, zijn er toch een paar andere pogingen gedaan om een althans iets bevredigender resultaat te verkrijgen.

Het eerste voorstel is Louis T. Milič met zijn poëzieprogramma RETURNER. Dit programma vertoont de volgende kenmerken.

Wat de input betreft kiest het programma niet uit een woordenboek maar uit een poëtisch universum, dat wil zeggen uit een langer gedicht.

RETURNER is geschreven in SNOBOL4. Milič begon zijn werk aan RETURNER in september 1969 en was klaar in december 1970. Het programma heeft 284 opdrachten in SNOBOL4. Het liep op een IBM 350/360. Het produceert gedichtstrofen. Per strofe heeft het programma zes seconden nodig.

Milič was de eerste die erop wees dat het namaken van computergedichten best een serieuze bezigheid zou kunnen zijn. Je zou er namelijk mee kunnen ontdekken wat de aard van de beperkingen is die poëtisch taalgebruik met zich meebrengt, maar ook wat de relatieve ingewikkeldheid van iedere afzonderlijke factor is. Milič zelf wilde met zijn RETURNER programma echter een bescheidener doel dienen, namelijk ten eerste een gedicht analyseren, en dan via de gevonden structuur en de gevonden woordenschat gedichten schrijven die lijken op het origineel. Daarbij verwachtte hij op een serie basisproblemen van taalsynthese te stoten.

- Het lexicon van Milič

Het lexicon werd afgeleid uit het gedicht zelf. Niet door er simpelweg een scanner op los te laten, maar ook voorzetselgroepen (bijvoorbeeld 'van drempel tot drempel') werden opgenomen in het woordenboek. Natuurlijk moest om dat te kunnen eerst het geheel van woordklaslabels worden voorzien. Milič deed dat met de hand. Met de kennis van de vorige paragraaf kan labelling van woordklassen tegenwoordig automatisch gebeuren.

De syntactische structuur was vastgelegd in een soort basisschema, als volgt: bijwoord - onderwerp - modificaties - werkwoord - bijwoord. Hij kortte zelf de structuur af met FSMVE. F, M en E zijn bijwoordgroepen. Deze groepen mogen ook leeg zijn.

Een klein probleem was natuurlijk ook nog de vormleer. Er moet in een dergelijk programma natuurlijk ook een stukje code zitten dat 'ik ben' tot 'hij is' kan omvormen als de nieuwe zin dat vereist.

Een beetje dobbelsteen zit ook in het programma om aan nieuwe ordeningen van het materiaal te kunnen komen. Hier werd een random generator gebruikt, omdat anders iedere keer het hele lexicon opnieuw gesorteerd zou moeten worden.

- Wat levert het programma op?

Doel van het programma was de nabootsing van een bepaald gedicht, althans de individuele strofen ervan. Als input gebruikte Milič het gedicht RETURN van Alberta Turner waarvan de tekst nu volgt.

RETURN by Alberta Turner

Hemlocks are nearly round,
Deer paw the pond,
My dog squirts the porch post.

Last night the snow wouldn't take tracks,
But apple twigs are cut
Higher than porcupines.

Yesterday I saw the weathercock
Through the willow;
Today the cock is gone.

Holding my bowl,
I step carefully
From salad to salad.

The swamp has shores again
And the quicksand grass.

Have I planted crowbars under my porch
And chisels under my rocker?

Noc rumbs fall from the agate pebble,
But around it acres of sand are also red.

Melons crack,
Locusts have shelled my sisters on the porch,
The collie's tongue sticks.

The kittens hiss, at the milk pan,
And knead separate blankets.

This morning all the apples ringed the tree
So close the boy turned his ankle
And rabbits staggered.

The jack o'lantern's soft now
And nearly gray.

Mica silvers the sheep.
When my child brings his paper star,
Will it glitter? It's not silver paper.

Hemlocks are nearly round,
Mice run under the snow,
Sunrise reddens a thin curd.

Een keuze uit de produktie zijn de volgende strofen:

In the morning crowbars will be nearly round,
Separate blankets never step again.
Tomorrow I will ring him through the willows.

Do mice sometimes become like deer at home?
Hemlocks hiss from salad to salad now
But yesterday he often pawed all the apples at the milk pan.

In the morning the quicksand will seem silver.
A thin curd also staggers again.

At home the weatherlock never appears like the boy
Yet does my bowl turn from salad to salad now?
Carefully we sometimes paw the porch post at the milk pan.

In the morning locusts will turn nearly gray,
Apple twigs often crack again,
So last night they took them through the willows.

Are the kittens also like my dog at home?
Pocupines fall from salad to salad now.
Yesterday I never saw his ankle at the milk pan.

Een deel van het RETURNER programma is als volgt:

```
      QM = '?'
      PL = '+'
      CO = '!'
      PU = PT | QM
      PLP = PL SP PL | PL PL
      LQ = 1
      LINE = SP
      RANDOM = '10123101432111320101114230111032114C'
      BK5 = '     '
      BK5C = '                                                  '
      BK6 = '      '
      BK = '                                    '
      NULL =
      NOCONTS = POS(0) NULL RPOS (0)
      VOWEL = 'AEIOU'
      BECOME = 'BECAME'
      FALL = 'FELL'
      STICK = 'STUCK'
      BRING = 'BROUGHT'
      SEE = 'SAW'
      BECAME = 'BECOME'
      TAKE = 'TOOK'
      TOOK = 'TAKE'
      FELL = 'FALL'
      STUCK = 'STICK'
      BROUGHT = 'BRING'
      SAW = 'SEE'
      P = 'ADVPHR,O,ADV,'
      S = 'NOUN,NOM,PRONS,'
      N = 'O,NOO,'
      V = 'VB,VI,VT,'
      P = 'O,END,ADVPHR,'
      O = 'PRONO,NOUN,NOM,'
      C = 'ADJ,NOUN,'
      L = 'O,LINK,O,O,O,LINK,O,O,LINK,LINK,O,'
      ADV = '8TOMORROW,8YESTERDAY,3TODAY,8CAREFULLY,8LAST-NIGHT,'
+     '8YESTERDAY,'
      ADVPHR = '8IN-THE-MORNING,8THROUGH-THE-WILLOWS,8AT-HOME,'
+     '8FROM-SALAD-TO-SALAD,8AT-THE-MILK-PAN,'
      PRONS = 'I,4ME,4SHE,HE,THEY,'
      NOUN = '5CROWBARS,4MY-DOG,5THE-KITTENS,5MELONS,5LOCUSTS,'
+                    '4THE TREE,4THE-BOY,4SUNRISE,4THE-WEATHERCOCK,'
+     '4THE SWAMP,4THE-QUICKSAND,4THE-POND,5DEER,4THE-SNOW,5MICE,'
      NOM = '5SEPARATE-BLANKETS,5HENLOCKS,5ALL-THE-APPLES,4A-TAIN-CURD,'
+     '4MY-BOWL,4THE-PORCH-POST,5APPLE-TWIGS,5PORCUPINES,4HIS-ANKLE,'
+     '4GRASS,5CHISELS,5NO-CRUMBS,5RABBITS,4MICA,4MY-CHILD,'
+     '4THE-AGATE-PEBBLE,5ACRES-OF-SAND,5MY-SISTERS,'
+     '4THE-COLLIE*S-TONGUE,'
      END = 'AGAIN,NOW,'
      NOO = 'NEVER,SOMETIMES,OFTEN,ALSO,'
      VB = 'ORE,1BECOME,1SEEN,1APPEAR,1TURN,'
      VI = '2STEP,2HISS,2STAGGER,2TURN,2CRACK,2FALL,2STICK,'
+     '2GLITTER,'
      VT = '3RING,3PAW,3SQUIRT,3PLANT,3SHELL,3KNEAC,3SEE,3BRING,'
+     '3TAKE,'
```

```
      PRONO = '4HIM,ME,THEM,US,HER,'
      ADJ = 'NEARLY-ROUND,SILVER,NEARLY-GRAY,SOFT,'
+     'SO-CLOSE,ALSO-RED,'
      LINK = 'BUT,YET,SO,AND,'
      R = 'OIS' | 'OAM' | 'OARE'
      P1 = ' I '
      P3 = '4HE ' | '4SHE '
      P1P = ' WE '
      P3P = ' THEY '
      PSP = P1 | P3 | P1P | P3P
      PP = P1 | P1P | P3P
      POB = 'ME' | 'THEM' | 'US' | 'HER' | 'HIM'
      PADV = 'YESTERDAY' | 'LAST-NIGHT'
      FADV = 'TOMORROW' | 'IN-THE-MORNING'
      VV = VB | VI | VT
      LK = ' AND ' | ' BUT ' | ' SO ' | ' YET '
*
*     FUNCTIONS
*
      DEFINE('SHUFFLE(SET)SET1,SET2,PIECE','SHU')
      DEFINE('STANZA(LINES)Z','STZ')
      DEFINE('TEST(STG)W','TST')                    :(START)
SHU   SET BREAK(CM) . PIECE CM =                      :F(SH2)
      SET1 = SET1 PIECE CM
      SET BREAK(CM) . PIECE CM =                      :F(SH2)
      SET2 = PIECE CM SET2                            :(SHU)
SH2   SHUFFLE = SET2 SET1                             :(RETURN)
STZ   LINES BREAK(SP) . FST
      LINEZ = LINES
ST2   LINEZ BREAK(SP) . WOD SP =                    :F(ST3)
      WOD PU                                        :S(ST4)
      STANZA = STANZA WOD SP
      Z = Z + 1
      EQ(Z,SIZE(FST))                               :F(ST2)
      OUTPUT = STANZA; STANZA =; Z =
      LINES = LINEZ                                 :(STZ)
ST3   OUTPUT = STANZA; OUTPUT =                     :(RETURN)
ST4   STANZA = STANZA WOD SP
      OUTPUT = STANZA; STANZA =; Z =
      LINES = LINEZ                           :(STZ)
TST   STG BREAK(SP) SP =                        :F(T2)
      W = W + 1                                :(TST)
T2    W RTAB(1) LEN(1) . DIG
      DIG ANY('13579')                        :S(RETURN)F(FRETURN)
*
*     PROGRAM 'START'--STRUCTURE BUILDING
*
START STRING = 'FSMVE'
      SENT = STRING
PICK  SENT LEN(1) . SLOT =                          :F(ADJ)
VERB  SLOT 'V'                                      :S(VLOOP)
LOOP   $SLOT BREAK(CM) . STRUC CM =
       $SLOT =  $SLOT STRUC CM
      INTEGER(STRUC)                                :S(PICK)
FILL   $STRUC BREAK(CM) . LEX CM =
```

```
        $STRUC = $STRUC LEX CM
        SLOT 'S'                                    :S(SLOOP)
        SLOT 'F'                                    :S(FLOOP)
        SLOT 'E'                                    :F(LINE)
        ESLOT = LEX                                 :(LINE)
FLOOP   FSLOT = LEX
LINE    LINE = LINE LEX SP                          :(PICK)
SLOOP   LINE = LINE '6' LEX SP                      :(PICK)
VLOOP    $SLOT BREAK(CM) . VERB CM =
         $SLOT =  $SLOT VERB CM
VB       VERB BREAK(CM) . LEX CM =
         $VERB =  $VERB LEX CM
        LINE = LINE LEX SP
        LEX ANY('01')                               :S(COMP)
        LEX '2'                                    :F(OBJ)
        TEST(LINE)                                 :S(PICK)
        ADVPHR BREAK(CM) . AP CM =
        ADVPHR = ADVPHR AP CM
        LINE = LINE '7' AP SP                       :(PICK)
OBJ     TEST(LINE)                                  :S(OBJ2)
        NOM BREAK(CM) . DO CM =
        NOM = NOM DO CM
        LINE = LINE '7' DO SP                       :(PICK)
OBJ2    PRONO BREAK(CM) . PO CM =
        PRONO = PRONO PO CM
        LINE = LINE '7' PO SP                       :(PICK)
COMP    TEST(LINE)                        :S(COMP2)
        ADJ BREAK(CM) . PA CM =
        ADJ = ADJ PA CM
        LINE = LINE '7' PA SP                       :(PICK)
COMP2    NOUN BREAK(CM) . PN CM =
        NOUN = NOUN PN CM
        LINE = LINE '7LIKE-' PN SP                  :(PICK)
```

3. Het filtermodel

De problemen van computerpoëzie zijn tweeërlei. Ten eerste is er het probleem van het schrijven van teksten in taal in het algemeen en ten tweede is er het probleem van de poëzie zelf. Het is duidelijk dat niet alles zomaar poëzie genoemd kan worden. Men kan wel alles eventueel gebruiken in een gedicht, maar op zich is een bepaald stukje taal nog geen poëzie.

Poëzie voortbrengen is een zo ingewikkelde zaak dat de inzichten voorshands ontbreken hoe het schrijven van gedichten als proces plaatsvindt. Dat betekent dat de computer voorlopig niet echt kan worden ingezet om 'vanuit het niets' gedichten te schrijven. Nu is het natuurlijk maar zeer de vraag of dichters wel zo vanuit het niets schrijven. Het tegendeel is eerder waar. Het lijkt wel of alles uit de lucht gegrepen wordt, maar dat kan niet echt zo zijn. Ten eerste heeft een beetje behoorlijk dichter zeer wel zijn taal geleerd. Ten tweede blijkt het altijd zo te zijn dat dichters van enige faam steeds om dezelfde thematieken en technieken heendraaien. Ze schrijven als het ware voortdurend variaties op een grondthema. Dat is een van de redenen waarom men dichters van enig niveau herkent als men kenner van poëzie is.

Daaruit zou men de conclusie kunnen trekken dat dichters met een zeer specifieke oogopslag naar de werkelijkheid om zich heen kijken. Dat ze die als het ware filtreren. Ze hebben bovendien hun eigen 'taaltuin', om het zo maar eens te zeggen, hun eigen specifieke woordgebruik en hun eigen techniek.

Uit deze ideeën en observaties is een experiment ontstaan dat als doel had een bepaald soort poëzie zo te schrijven dat het voor de kenner als dat soort poëzie herkenbaar was. Dit experiment werd opgezet voor een manifestatie 'Kunst en Computer' die in 1981 in Den Haag werd gehouden onder het aegis van de Haagse Kunstkring.

Het was de bedoeling met de computer vrije verzen te maken die konden doorgaan voor 'moderne' poëzie. Als techniek werd gekozen de herhaling, een zeer veel voorkomende techniek in alle soorten poëzie. Als filter werd gekozen: kleuraanduidingen. Op deze manier doet de computer via patroonherkenning wat de dichter doet via zijn manier van kijken of 'filteren', dat wil zeggen de computer bootst de dichter na, althans wat dit aspect betreft.

Als databestand werd gekozen de verzamelde gedichten van A. Roland Holst, Lucebert en Martin Boot.

Als programma's werden gebruikt de programma-bibliotheek LIBOLIB, een programmabibliotheek die vergelijkbaar is met OCP en een extra programma voor het definitieve printwerk. Het geheel werd genoemd BORANPO. BORANPO leverde onder andere de volgende teksten voor twee stemmen op:

BORANPO 3^{b1} - Computational variations on poetry
by Martin Boot

DE DRAAD VAN DE DAUW DIE JE VASTHIELD VAN DREMPEL TOT DREMPEL
DE DRAAD VAN DE DAUW DIE JE VASTHIELD VAN DREMPEL TOT DREMPEL

GEPIKKELDE STERREN VALLEN NIET VER VAN EEN HARIGE HUID

ZIJ ALLEN STAAN STIL IN DE SNEEUW BIJ WAT SPINSEL
EN RAVEN DOEN WANDELEN ONDER DE SNEEUW

DE DRAAD VAN DE DAUW DIE JE VASTHIELD VAN DREMPEL TOT DREMPEL
DE DRAAD VAN DE DAUW DIE JE VASTHIELD VAN DREMPEL TOT DREMPEL

GEPIKKELDE STERREN VALLEN NIET VER VAN EEN HARIGE HUID
PAK JE JAS NU JE HUID IS TE KLEIN

GEPIKKELDE STERREN VALLEN NIET VER VAN EEN HARIGE HUID
EN RAVEN DOEN WANDELEN ONDER DE SNEEUW
DE SNEEUW UIT JE ZWEVEND GEBUNDELDE VATEN
GEJAAGDEN WANDELEN SNEEUW

DE DRAAD VAN DE DAUW DIE JE VASTHIELD VAN DREMPEL TOT DREMPEL
DE DRAAD VAN DE DAUW DIE JE VASTHIELD VAN DREMPEL TOT DREMPEL

VAN DREMPEL TOT DREMPEL
VAN DREMPEL TOT DREMPEL

EEN GOUDEN DIE DOORSTROOMT

GEPIKKELDE STERRREN VALLEN NIET VER VAN EEN HARIGE HUID
PAK JE JAS NU JE HUID IS TE KLEIN
DE HUID VAN EEN VIS

ZIJ ALLEN STAAN STIL IN DE SNEEUW BIJ WAT SPINSEL
EN RAVEN DOEN WANDELEN ONDER DE SNEEUW
DE SNEEUW UIT JE ZWEVEND GEBUNDELDE VATEN
GEJAAGDEN WANDELEN SNEEUW
EN SNEEUW OP DE KOELTE VOOR ALLE
SPIEGELEN GYNEKOLOGEN DE WUIVENDE SNEEUW

DE DRAAD VAN DE DAUW DIE JE VASTHIELD VAN DREMPEL TOT DREMPEL
DE DRAAD VAN DE DAUW DIE JE VASTHIELD VAN DREMPEL TOT DREMPEL
VAN DREMPEL TOT DREMPEL
VAN DREMPEL TOT DREMPEL

EEN GOUDEN DIE DOORSTROOMT

GEPIKKELDE STERREN VALLEN NIET VER VAN EEN HARIGE HUID
DE HUID VAN EEN VIS

ZIJ ALLEN STAAN STIL IN DE SNEEUW BIJ WAT SPINSEL
EN RAVEN DOEN WANDELEN ONDER DE SNEEUW
DE SNEEUW UIT JE ZWEVEND GEBUNDELDE VATEN
GEJAAGDEN WANDELEN SNEEUW
EN SNEEUW OP DE KOELTE VOOR ALLE
SPIEGELEN GYNEKOLOGEN DE WUIVENDE SNEEUW

DE DRAAD VAN DE DAUW DIE JE VASTHIELD VAN DREMPEL TOT DREMPEL
DE DRAAD VAN DE DAUW DIE JE VASTHIELD VAN DREMPEL TOT DREMPEL
VAN DREMPEL TOT DREMPEL
VAN DREMPEL TOT DREMPEL

EEN GOUDEN DIE DOORSTROOMT

GEPIKKELDE STERREN VALLEN NIET VER VAN EEN HARIGE HUID
PAK JE JAS NU JE HUID IS TE KLEIN
DE HUID VAN EEN VIS

ZIJ ALLEN STAAN STIL IN DE SNEEUW BIJ WAT SPINSEL
EN RAVEN DOEN WANDELEN ONDER DE SNEEUW
DE SNEEUW UIT JE ZWEVEND GEBUNDELDE VATEN
GEJAAGDEN WANDELEN SNEEUW
EN SNEEUW OP DE KOELTE VOOR ALLE
SPIEGELEN GYNEKOLOGEN DE WUIVENDE SNEEUW
DE HOLLENDE SNEEUW EN DE LICHTE DAG

BORANPO 3 (gedeeltelijk)

NIET DE NONNEN VAN MEDICIJNEN NIET DE KNAPEN UIT HET BLONDE

SNOEPEN UW LIPPEN GIFTIGE BESSEN UIT GROENE DAGBOEKBLADEN
WANT DOOR DE GROENE DAGBOEKBLADEN VRETEN

NOG ALTIJD BEZIT IK MIJN WITTE EN RODE PENNEN WAARUIT DE LOTUS EN

EN DE VLAMMEN EN HET ROOD EN ALS DE TUIMEL LIEFSTEWRAKEN

JE ZIET IK DRAAG HET HANDGRANAAT EN GAS NIET WIT OF ZWART NIET

DAN STOND DE RIJST OP VAN JE NIEUW WITTE HUID
TOEN MET WITTE LUCHTAPEN DE MAAN OPTROK HEELAL VAN LEGERS
WITTE METEOREN
NOG ALTIJD BEZIT IK MIJN WITTE EN RODE PENNEN WAARUIT DE LOTUS EN

JE ZIET IK DRAAG HET HANDGRANAAT EN GAS NIET WIT OF ZWART NIET

VOOR KIL HET VERSTEENDE DAGEGRAAN DAT BROKKELT UIT DE ZWARTE

NIET DE NONNEN VAN MEDICIJNEN NIET DE KNAPEN UIT HET BLONDE

SNOEPEN UW LIPPEN GIFTIGE BESSEN UIT GROENE DAGBOEKBLADEN
WANT DOOR DE GROENE DAGBOEKBLADEN VRETEN

NOG ALTIJD BEZIT IK MIJN WITTE EN RODE PENNEN WAARUIT DE LOTUS EN

EN DE VLAMMEN EN HET ROOD EN ALS DE TUIMEL LIEFSTEWRAKEN
JE ZIET IK DRAAG HET HANDGRANAAT NIET WIT OF ZWART NIET

DAN STOND DE RIJST OP VAN JE NIEUWE WITTE HUID
TOEN MET WITTE LUCHTAPEN DE MAAN OPTROK HEELAL VAN LEGERS
WITTE METEOREN
NOG ALTIJD BEZIT IK MIJN WITTE EN RODE PENNEN WAARUIT DE LOTUS EN

JE ZIET IK DRAAG HET HANDGRANAAT EN GAS NIET WIT OF ZWART NIET
ZWART ONDER DE TAFELS VERKOOLD

VOOR KIL HET VERSTEENDE DAGEGRAAN DAT BROKKELT UIT DE ZWARTE

EEN BLAUW EN STRAKGESPANNEN LIJF

NIET DE NONNEN VAN MEDICIJNEN NIET DE KNAPEN MET HET BLONDE
DE BLONDE ORGELS DER WOLKEN OP DE HORIZON

SNOEPEN UW LIPPEN GIFTIGE BESSEN UIT GROENE DAGBOEKBLADEN
WANT DOOR DE GROENE DAGBOEKBLADEN VRETEN
DOOR DIE GROENE OF MOEDE
HET GROENE JUBELEN VAN HET VOORJAAR

NOG ALTIJD BEZIT IK MIJN WITTE EN RODE PENNEN WAARUIT DE LOTUS EN

EN DE VLAMMEN EN HET ROOD EN ALS DE TUIMEL LIEFSTEWRAKEN
ALS ROOD VERDORDE SNOEKEN

JE ZIET IK DRAAG HET HANDGRANAAT EN GAS NIET WIT OF ZWART NIET
IK DACHT DAT CHRISTUS WIT WAS
IK DACHT DAT HIJ ONS WIT ZOU ZITTEN

DAN STOND DE RIJST OP VAN JE NIEUWE WITTE HUID
TOEN MET WITTE LUCHTAPEN DE MAAN OPTROK HEELAL VAN LEGERS
WITTE METEOREN
NOG ALTIJD BEZIT IK MIJN WITTE EN RODE PENNEN WAARUIT DE LOTUS EN
DUIZELENDE POLYDAUKES WITTE HYLOS

JE ZIET IK DRAAG HET HANDGRANAAT EN GAS NIET WIT OF ZWART NIET
ZWART ONDER DE TAFELS VERKOOLD
ZWART EEN NIET TE KENNEN KLEUR
VOOR KIL HET VERSTEENDE DAGEGRAAN DAT BROKKELT UIT DE ZWARTE
ZIJN IN KOELE ZWARTE TAKKETORENS EEN GLOEIENDE APPEL
EN HEEFT MET ZWARTE HANDEN VERDEELD

LITERATUUR

Allen, J.: "Anatomy of LISP", McGraw-Hill Book Cy., New York, 1978.

Aho, A. & J. Ullman: "The Theory of Parsing, Translation and Compiling", Prentice Hall, Englewood Cliffs, N.J., 1972.

Arbib, M.A.: "Computers en de Cybernetische Samenleving", Academic Service, Den Haag, 1978.

Berkeley, E.C. & D.G. Bobrow (eds.): "The programming Language LISP: its Operation and Applications", The MIT Press, Cambridge, Mass., 1974.

Boole, G.: "The Laws of Thought", Dover, New York, 1958.

Boot, M., E. Maat & J. Renkers: "A Model for Automated Phonemization", Utrecht, 1980.

Boot, M.: Automated Lemmatization of written Dutch, "ALLC Bulletin", 1980.

Boot, M.: "Homographie: Ein Beitrag zur automatischen Wortklassenzuweisung in der Computerlinguistik", Utrecht, 1979.

Boot, M.: Linguistic Data structures, Reducing Encoding by hand and Programming Languages. In: "The Computer in Linguistic and Literary Studies", A. Jones & R.F. Churchhouse (eds.), Cardiff, 1976.

Boot, M.: "LIBOLIB, I: Technieken en Programma's voor Linguistische en Literaire Standaardbibliotheken", Utrecht, 1976.

Bolt, Beranek, Newman Inc.: "Speech Understanding Systems", BBN Report 3438, 1976.

Bobrow, D., R. Kaplan, M. Kay, D. Norman, H. Tompson & T. Winograd: "GUS: A Frame-Driven Dialog System", Artificial Intelligence, North Holland, Amsterdam, 1977.

Bobrow, D.: The LISP Program for METEOR. In: "The Programming Language LISP", Berkeley, E.C. & D. Bobrow (eds.), MIT Press, pp. 249-259, 1974 (4e druk).

Charniak, E. & Y. Wilks (eds.): "An Introduction to Artificial Intelligence and Natural Language Comprehension", North Holland, Amer. Elsevier, Amsterdam, 1975.

Charniak, E. et al.: "Artificial Intelligence Programming", Lawrence Erlbaum Ass., Hillsdale, New Jersey, 1980.

Conway, J.H.: "Regular Algebra and Finite Machines", Chapman and Hall, London, 1971.

Duda, R.O. & J.V. Gasching: Knowledge Based Expert Systems Come of Age, "Byte", vol. 6, nr. 9, pp. 238-281, 1981.

Foster, J.M.: "List Processing", MacDonald/Elsevier, London, 1967.

Ginsburg, S.: "Formal Languages", North Holland, Amsterdam, 1975.

Heer, T. de: "Experiments with multi-lingual queries in a multilingual data-base", Mededeling van het TNO voor Wiskunde, informatica en statistiek, Den Haag, 1978.

Hennie, F.C.: "Finite-State Models for Logical Machines", Wiley & Sons, New York, 1968.

Hofstadter, D.R.: "Gödel, Escher, Bach, An Eternal Golden Braid", The Harvester Press, Sussex, England, 1979.

Hunt, E.B.: "Artificial Intelligence", Academic Press, New York, 1975.

Iverson, K.: "APL Language", IBM Corporation, P.O.Box 50020, San Jose, California, Ca95150, 1978.

Kickert, W.J.M.: "Fuzzy theories on decision making", Martinus Nijhoff, Leiden, 1978.

Klein, S. et al.: "Automatic Novel Writing: A Status Report", Madison, 1973.

Knuth, D.E.: "The Art of Computer Programming", vol.1, 2, 3, Addison-Wesley, Reading, Mass., 1969.

Koppelaar, H. et al.: Verbale Modelvorming, "Sociologische Gids", 25, Boom, Meppel, pp.201-211, 1978.

Koppelaar, H.: "Linguistic Modelling with APL. In: APL '80, G. van der Linden (ed.), North Holland Publ. Cy., 1980.

Lakoff, G.: Hedges: A Study of fuzzy concepts. In: "Proc. 8th Meeting Chicago Linguistic Society", Univ. of Chicago Linguistic Dept., 1972.

Lehnert, W.: "The Process of Question Answering: A Computer Simulation of Cognition", Hillsdale, 1978.

Lukasiewicz, J.: Philosophical remarks on many-valued systems of propositional logic. In: "Polish Logic 1920-1939", McCall, S. (ed.), Clarendon Press, Oxford, 1967.

Marcus, S.: "Algebraic Linguistics, Analytical Models", Academic Press, New York, 1967.

Meehan, J.R.: "The Metanovel: Writing Stories by Computer", New York, 1980.

Minsky, M. (ed.): "Semantic Information Processing", MIT Press, 1968.

Nash-Webber, B.: An AI Approach to Mantra Generation, "SIGART Newletter", nr.47, pp.17, 1974.

Negoita, C.V. & D.A. Ralescu: "Application of Fuzzy Sets to Systems Analysis", Birkhäuser Verlag, Basel, 1975.

Newell, A. & H. Simon: "Human Problem Solving", Pretence Hall, Englewood Cliffs, N.J., 1972.

Pivar, M. et al.: The LISP Programs for Inductive Inference on Sequences. In: Berkeley E.C. & D. Bobrow (eds.): "The Programming Language LISP", MIT Press, pp.260-289, 1974.

Proceedings on Text Manipulation, "Sigplan Notices", vol.16, nr.6, ACM bestelnr.548810, P.O.Box 64145, Baltimore MD 21264, 1981.

Renkers, J. & M. Boot: "FONGRAF: A Computer program for automated Phonemization", Utrecht, 1980.

Rieger, Ch.: "Conceptual Memory: A Theory and Computer Program for Processing the Meaning Content of Natural Language Utterances", Stanford, 1974.

Schank, R.C. & K.M. Colby: "Computer Models of Thought and Language", San Francisco, 1973.

Schank, R.C. & R. Abelson: "Scripts Plans Goals and Understanding: An Inquiry into Human Knowledge Structures", Hillsdale, 1977.

Schank, R.C.: "Conceptual Information Processing", North Holland, Elsevier, Amsterdam, 1975.

Siklóssy, L.: "Let's Talk LISP", Prentice Hall, Inc., Englewood Cliffs, New Jersey, 1976.

Starke, P.H.: "Abstract Automata", North Holland, Amsterdam, 1972.

Teitelman, W.: "InterLISP reference manual", Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, California, 1974.

Waterman, D.A. & F.H. Hayes-Roth (eds.): "Pattern-Directed Inference Systems", New York, 1978.

Weizenbaum, J.: "Computer Power and Human Reasoning", San Francisco, 1975.

Wilensky, R.: "Understanding Goal Based Stories", New York, 1980.

Wilks, Y.A.: "Grammar, Meaning and the Machine Analysis of Language", Rountledge & Kegan Paul, London, 1972.

Winograd, T.: "Procedures as a Representation for Data in a Computer Program for Understanding Natural Language", MIT Press, Cambridge, Mass., 1970.

Winograd, T.: "Understanding Natural Language", Edinburg, 1972.

Winston, P.H.: "Artificial Intelligence", Addison-Wesley Publ. Cy., Reading, Mass., 1977.

Winston, P.H. & B.K.P. Horn: "LISP", Addison-Wesley Publ. Cy., Reading, Mass., 1981

Woods, W.: "The Lunar Sciences Natural Language System", 1972.

Woods, W.: "Semantics for a Question Answering System", New York, 1979.

Zadeh, L.A.: Fuzzy Sets, "Information and Control", 8, pp. 338-353, 1965.

Zadeh, L.A.: A Fuzzy-set-theoretical interpretation of linguistic hedges, "Journal of Cybernetics", 5, pp. 4-34, 1972.

Zadeh, L.A., The concept of a linguistic variable and its application to approximate reasoning II, "Information Sciences", 8, pp. 301-357, 1975.

Zampolli, A. (ed.): "Linguistic Structures Processing", New York, 1977.

APPENDIX BIJ HOOFDSTUK 2

```
BOOIND:
        PROC OPTIONS(MAIN);
  PRINT:
         PROCEDURE (I) RECURSIVE;
         DCL W PTR, I FIXED BIN;
         IF LINKS(I) > 0 THEN CALL PRINT(LINKS(I));
         W=P(I);    PUT       EDIT(W->IMAGE.WORD,W->IMAGE.TOT,
            W->IMAGE.REG,W->IMAGE.PL)    (A,X(5),F(3),X(5),
                      2(F(3),X(2)));
         DO WHILE (W->IMAGE.GLK ¬= NULL);
              W=W->IMAGE.GLK;
              PUT EDIT(W->IMAGE.REG,W->IMAGE.PL)
               ( X(4),2(F(3),X(2)));
         END;
         PUT SKIP;
         IF RECHTS(I)> 0 THEN CALL PRINT(RECHTS(I));
         END PRINT;
   DCL
         NULL BUILTIN,
       CARD CHAR(80),
          (KTR,IB) FIXED BIN STATIC EXTERNAL,
       KADO CHAR(4),
      /* KODE NODIG VOOR KONKORDANTIE OP HAVO EXAMENS */
       CARDA(80) CHAR(1) DEF CARD,
       KARTA(80) CHAR(1) UNALIGNED,
       KART CHAR(80) DEF KARTA,
      WO CHAR(20),
          IR FIXED BIN STATIC EXTERNAL,
       S POINTER STATIC EXTERNAL,
       1 IMAGE BASED(S),
        2 WORD CHAR(20),
        2 ((REG,PL,TOT) FIXED BIN),
        2 GLK PTR,
      P(2000) POINTER STATIC EXTERNAL,
      ((RECHTS,LINKS)(2000)) UNALIGNED FIXED BIN,
     L2 FIXED BIN STATIC EXTERNAL,
    IPS FIXED BIN STATIC EXTERNAL,
      T POINTER STATIC EXTERNAL;
         ON ENDFILE(INTEX) BEGIN;
                                XS=1;
                                GOTO SORT;
                           END;
         OPEN FILE(INTEX) STREAM INPUT LINESIZE (80);
         IPS=2000;
         DO I= 1 TO IPS;
              P(I)=NULL;    RECHTS(I),LINKS(I)=0;
         END;
         XS=0;
         IA=0;
         CALL INFIND;
 NU:
         IP=81;
         WO=' ';
         IR,LP=0;
```

```
IN:
      IF IR=IPS THEN GOTO SORT;
 NI:
      GET FILE (INTEX )  EDIT (CARD )    (A(80));
      IF CARD=(80)' ' THEN GOTO SORT;
      IF CARDA(80)=' ' THEN DO;
      IA=IA+1;
      LP=0;
                              END;
      DO N=1 TO 80;
           IF CARDA(N)>='A' & CARDA(N) <='Z' THEN
              KARTA(N)=CARDA(N);
                                                ELSE KARTA(N)=' ';
      END;
      KARTA(80)=' ';
      I=INDEX(KART,' ');
      IF I< 2 THEN GOTO VUL;
      L=1;
      IB=I-1;
IN:
      IF IR=IPS THEN GOTO SORT;
 NI:
      GET FILE (INTEX )  EDIT (CARD )    (A(80));
      IF CARD=(80)' ' THEN GOTO SORT;
      IF CARDA(80)=' ' THEN DO;
      IA=IA+1;
      LP=0;
                              END;
      DO N=1 TO 80;
           IF CARDA(N)>='A' & CARDA(N) <='Z' THEN
              KARTA(N)=CARDA(N);
                                                ELSE KARTA(N)=' ';
      END;
      KARTA(80)=' ';
      I=INDEX(KART,' ');
      IF I< 2 THEN GOTO VUL;
      L=1;
      IB=I-1;
      WO=SUBSTR(KART,L,IB);
      IF WO=' ' THEN GOTO VUL;
 LUV:
      IR=IR+1;
      ALLOCATE IMAGE;
      S->IMAGE.WORD=WO;
      S->IMAGE.REG=IA;
      LP=LP+1;
      S->IMAGE.PL=LP;
      S->IMAGE.TOT=1;
      P(IR)=S;
      S->IMAGE.GLK=NULL;
      IF IR=IPS THEN GOTO SORT;
VUL:
      L=I+1;                  M=IP-L;
      IF M<1 THEN GOTO UIT;
      KART=SUBSTR(KART,L,M);
      I=INDEX(KART,' ');
      IF I=1 THEN DO;
```

```
                    IF M>3 THEN DO;
                    KADO=SUBSTR(KART,1,4);
                    IF KADO='    ' THEN GOTO IN;
                                    END;
                    GOTO VUL;
                  END;
      IB=I-1;
      WO=SUBSTR(KART,1,IB);
VUL1:
      IR=IR+1;
      ALLOCATE IMAGE;
      S->IMAGE.WORD=WO;
      S->IMAGE.REG=IA;
      LP=LP+1;
      S->IMAGE.PL=LP;
      S->IMAGE.TOT=1;
      S->IMAGE.GLK=NULL;
      P(IR)=S;
      IF IR=IPS THEN GOTO SORT;
      GOTO VUL;
UIT:
      IF XS=0 THEN GOTO IN;
SORT:
      IF IR > 0 THEN DO;
  DO I= 2 TO IR;
      L2=1;
  VGL:
      IF P(I)->IMAGE.WORD > P(L2)->IMAGE.WORD THEN GOTO GR;
      IF P(I)->IMAGE.WORD < P(L2)->IMAGE.WORD THEN GOTO KL;
      /* GELYK */
      P(L2)->IMAGE.TOT=P(L2)->IMAGE.TOT+1;
      IF P(L2)->IMAGE.GLK=NULL THEN DO;
                                  P(L2)->IMAGE.GLK=P(I);
                                    GOTO ENDLP;
                                  END;
                             ELSE DO;
                T=P(L2)->IMAGE.GLK;
           TEST:
                IF T->IMAGE.GLK=NULL THEN DO;
                          T->IMAGE.GLK=P(I);
                               GOTO ENDLP;
                                       END;
                T=T->IMAGE.GLK;
                GOTO TEST;
                               END;
  GR:
                   IF RECHTS(L2)=0 THEN DO;
                               RECHTS(L2)=I;
                         GOTO ENDLP;
                                       END;
                   L2=RECHTS(L2);
                   GOTO VGL;
KL:
                   IF  LINKS(L2)=0 THEN DO;
                                LINKS(L2)=I;
                               GOTO ENDLP;
                                       END;
                   L2=LINKS(L2);
                   GOTO VGL;
```

```
  ENDLP:
    END;
    AF:
         I=1;   CALL PRINT(I);
         DO I= 1 TO IR;
              FREE P(I)->IMAGE;
              P(I)=NULL;   RECHTS(I),LINKS(I)=0;
         END;
                         END;
         IR=0;
         IF XS=0 THEN GOTO NU;
END BOOIND;
```

SLEUTELS BIJ DE VRAGEN EN OPGAVEN

p. 21/22

1. Het programma in COBOL voor de standaardbrief bestaat uit drie functionele eenheden:
   1: de zogenaamde Data Division (13-30)
   2: de zogenaamde Working Storage Section (31-65)
   3: de Procedure Division (66-90)

2. De invoer wordt geregeld in de procedure division in regel 70.
   De invoer wordt gelezen volgens de specificaties in de data division 15-24.
   De uitvoer wordt geregeld in de procedure division 71-87. In deze opdrachten wordt verwezen naar de specificaties in de working storage section 32-64.

3. Het standaardgedeelte van de brief bevindt zich in de working storage section.

4. Spaties in een regel worden aangebracht door de opdracht in de working section:

   ```
   02 FILLER PIC X(n) VALUE SPACES
   ```

   waarbij n een reëel getal voorstelt.

5. Doorlopen van een programmaregel geeft men aan door een - teken te plaatsen vóór kolom 7 van de volgende kaart (zie bijvoorbeeld regel 51).

6. Het eind van een opdracht wordt in COBOL aangegeven door een punt ('.').

7. In regel 70 leest men wat het programma moet doen als er geen invoergegevens meer zijn. (AT END GO TO ...)

8. Interlinie wordt verzorgd door te manipuleren met spaties.

p. 51

1. Neen, want de regel is al volledig.

2. *GO.

3. *INPUT
   TEXT TO 50
   *GO

4. *ACTION
   DO CONCORDANCE
   *FORMAT
   KEYS LEFT SAME LINE

5. *INPUT
   TEXT TO 65. PROCESS FROM LINE 50 TO LINE 5000, WHERE
   A="VONDEL".
   *ACTION
   DO CONCORDANCE. PICK LENGTH LT 10, FREQUENCY GT 2.
   DEFINE "ZIJN" = "BEN, BENT, WAS, ZIJN, WAREN, WAART,
   GEWEEST". REFERENCES P=3, L=3.
   MAXIMUM CONTEXT LEFT 2 WORDS, RIGHT 2 WORDS.
   *FORMAT
   REFERENCE RIGHT. LAYOUT 2 COLUMNS BY 50 WIDE
   TITLE "EEN STUKJE VONDEL", PAGE 25.

p. 74

1. De uitspraken zijn strikt vraag/antwoord. De computer begint. Dus om en om zijn de regels van de computer.

2. Deze vraag zal door verschillende mensen verschillend beoordeeld worden, maar het is duidelijk dat op een gegeven moment niet meer duidelijk is wie wie is.

3. Zodra de geïnterviewde een vraag stelt aan de computer.

4. Dan kiest hij maar wat om verder te komen, ofwel: het onderwerp wordt plotseling veranderd.

p. 93

2a, 2d, 2e, 2f

p. 96

```
(DEF (VEEG (OUDFEIT)(SETQ DATA_BASE
        (EFFACE OUDFEIT DATA_BASE))))
```

p. 101

De laatste regel kan weg, op enkele sluithaken na. Immers ook als VEEG2 niets geveegd heeft, levert deze toch geen NIL, dus zal de CONDitie haar tweede argument (namelijk het herinstalleren van het gegevensbestand inclusief nieuwe feiten) altijd uitvoeren. Sterker nog, de VEEG2 taak zal nooit NIL leveren bij een bestaand gegevensbestand, dus is de laatste regel overbodig.

p. 116

```
(*  (:")      ( DAT )    /)
(*  ( SNEEUWT $ )      ( 2 1 )  *)
(*  ( IS $ )      ( 2 1 )  *)
(*  ( IK BEN $ HE )      ( ZIJ $ IS )  *)
(*  ( IK MOET $ GEBRUIKEN )      ( HIJ $ 2 3 )  *)
```

p. 118

```
(*  ( ($.1) )      ( (*E 1) )  *)
(*  ( T )          ( 0 )       /)
(*  ( D E )        ( 0 )       /)
(*  ( G E )        ( 0 )       /)
(*  ( D E N )      ( 0 )       /)
(*  ( E N )        ( 0 )       /)
```

p. 120

```
(   ( E R ($.0) )      ( 1 2 2 E T J E )    /)
(   ( E I ($.0) )      ( 1 1 T J E )        /)
(   ( M ($.0) )        ( 1 P J E )          /)
(   ( N ($.0) )        ( 1 1 E T J E )      /)
(   ( F ($.0) )        ( 1 J E )            /)
(   ( E I ($.0) )      ( 1 2 T J E )        /)
(   ( L ($.0) )        ( 1 T J E )          /)
(   ( ($.2) I N G ($.0) )      ( 1 2 3 K J E )    /)
```

p. 123

1. Zeker, denk aan woorden als 'staf', 'straf', 'staaf'. Zonder ($.0) zouden die woorden worden afgebroken als 'st-af', 'str-af', 'sta-af'.

2. Zeker, anders zou het losse woord 'af' een afbreekstreep krijgen.

4. 'af-rekenen', 'af-bekken',
   'A-frika', 'A-froaziatisch'.

p. 128

```
(*  ( DE $ VROUW )      ( (*K NP / 1 2 3 ) )    /)
```

p. 141

Nee, want ($.2) is één element voor de zoekprocedure, er kan dus niets tussen!

p. 151

Door eerst de klinkers te coderen

```
(*  ( E E )      ( 50 )   /)
(*  ( E )        ( 40 )   /)
```

p. 152

Anders wordt de 'N' herhaald!

p. 163

4.
```
(DEF (LAATSTE (X)(COND
        ((NULL X)   X)
        ((NULL (CDR X)) (CAR X))
        (T (LAATSTE (CDR X)))
))
```

5.
```
(DEF (OPEENNALAATSTE (X)(COND
        ((NULL X)   NIL)
        (NULL (CDR X)) NIL)
        ((EQN (LENGTH X) 2)(CAR X))
        (T (OPEENNALAATSTE (CDR X)))
))
```

Deze functie reageert op de lijst van VRIENDEN (SETQ VRIENDEN '(DONAHUE WIENER MCCULLOCH)) via de aanroep (OPEENNALAATSTE VRIENDEN) met het antwoord WIENER. Maar op de lijst van VIJANDEN (SETQ VIJANDEN '(XQUATL (VKWL ACTH) CPB)) komt er met (OPEENNALAATSTE VIJANDEN) de waarde (VKWL ACTH). Om hierin verbetering aan te brengen kan voor deze situatie in de definitie de (CAR X) vervangen worden door (LAATSTE X). Echter, dit is onvoldoende voor de situatie dat (((A, B, C))) aangeboden wordt. Om meer nestingen aan te kunnen moet de tweede regel van de definitie veranderen in ((NULL (CDR X))(OPEENNALAATSTE (CAR X))).

6. De functie die het N-de element van een lijst vindt is kortweg NTH. Deze zit standaard in LISP systemen. Dus (NTH 3 '(A B C)) levert C op. Nu moeten we even opletten. In wiskundige kringen begint tellen met nul, dus krijgen we (NTH 2 '(A B C)) C, dit laatste is ook te vinden in Pivars artikel over het aanvullen van letter- en getalreeksen (zie het boek van Berkeley & Bobrow). De definitie van NTH kan luiden:

```
(DEF (NTH (GETAL LIJST)(COND
        ((NULL LIJST) NIL)
        ((EQN GETAL 1)(CAR LIJST))
        (T (NTH (- GETAL 1)(CDR LIJST)))
)))
```

Hieruit is de definitie van de VINDNDECDR gemakkelijk af te leiden. Immers de opdracht (CAR LIJST) moet in essentie verdwijnen.

```
(DEF (VINDNDECDR (GETAL LIJST)(COND
        ((NULL LIJST) NIL)
        ((EQN GETAL 0) LIJST)
        (T (VINDNDECDR (- GETAL 1)(CDR LIJST)))
)))
```

7. De opdracht COPY is ingevoegd om ervoor te zorgen dat de oorspronkelijke versie bewaard blijft en REPLACE die niet definitief kan veranderen.

p. 169

1.
```
(DEF (UNION (X Y)(COND
        ((NULL X) Y)
        ((MEMBER (CAR X) Y)(UNION (CDR X) Y))
        (T CONS (CAR X)(UNION (CDR X) Y)))
)))
```

2.
```
(DEF (DOORSNEDE (X Y)(COND
        ((NULL X) NIL)
        ((NULL Y) NIL)
        ((MEMBER (CAR X) Y)(CONS CAR X)
                           (DOORSNEDE (CDR X) Y)))
        (T (DOORSNEDE (CDR X) Y))
)))
```

p. 173

1. verleden tijd
2. modaliteit
3. modaliteit
4. 'niet willen' vervalt; komt = stellend
5. stellend + negatief
6. vragend
7. (bepaling van) tijd

p. 174

1. Gaan, CA1 Jan, LOCUS naar bed
2. Gaan, CA1 Jan, CA2 Marie, LOCUS naar bed
3. Gaan, CA1 Jan, INSTR met de lift, LOCUS naar bed
4. Zitten, CA1 Marie, LOCUS in de tuin
5. Zitten, CA1 Marie, T1 met een breiwerkje, LOCUS in de tuin

p. 180

Het eerste toen is een voegwoord, het tweede toen is een bijwoord.

p. 186/187

1. De kennis van de wereld is de kennis die nodig is om te begrijpen wat groot en klein is, wat piramiden, kubussen en blokken zijn, wat kleuren zijn en wat het betekent dat iets zich in/op etc. iets anders bevindt. Kortom het moet de kennis zijn die nodig is om fysieke objecten te beschrijven en te hanteren.
2. a. Op zinsniveau moet de computer een zinsontleder hebben die bovendien semantische kennis heeft. De computer moet niet alleen weten of iets een grammaticaal woord of een inhoudswoord is, hij moet ook nog weten wat de inhoud van de inhoudswoorden is. Verder moet hij de onderlinge verbanden (syntaxis) van de woorden en inhouden binnen de zin weten.
   b. Op meer dan zinsniveau moet de computer in staat zijn verwijzende woorden te achterhalen. Hij moet kunnen vinden waar ze op slaan.
3. Er komen vragen in de dialoog voor die direct slaan op de wereld van de blokken.
   Er komen vragen in de dialoog voor die slaan op de kennis die gedurende de dialoog werd opgebouwd.
   Er komen vragen in de dialoog voor die slaan op de handelingen die in de dialoogsessie werden verricht.
4. Om vragen te beantwoorden moet de computer dus de kennis hebben die in vraag 2 werd gevraagd. Verder moet hij de kennis hebben die in vraag 1 werd gevraagd. Tot slot moet de computer weten hoe hij antwoorden formuleert in de taal waarin hij moet antwoorden.
5. Nee, de computer moet ook kennis kunnen opnemen. Verder moet hij gevolgtrekkingen kunnen maken. Ook moet hij de geschiedenis van zijn eigen dialoog kunnen terugvinden (onthouden).
6. Natuurlijk is het mogelijk ook vragen in het Nederlands te beantwoorden. Het enige dat daarvoor veranderd moet worden, is het output-gedeelte (zie de laatste zin van antwoord 4).

p. 200

Onder de eerste dubbele pijl, want het slaat op Kees en op slaan, die een dubbele relatie hebben (Kees is nodig om te slaan en Kees slaat). Vandaar de dubbele pijl!

p. 217

Ik denk me (te verdrinken)
Ik kan me (niet meer houden)
Ik wil (me niet langer laten uitbuiten)
enzovoorts.

Alleen 'hoop' en 'denk', want bij de woorden als 'wil' en 'zal' heeft het weinig zin te vragen: Waar komt die wil vandaan. Bij 'zal' is het misschien wel aardig te vragen: Waar komt deze toekomstverwachting bij je vandaan?

p. 236

De oplossing is al gegeven in het programma dat zinnen genereert.

ACADEMIC SERVICE INFORMATICA UITGAVEN

INLEIDINGEN

'Basiskennis Informatieverwerking' van Jan Everink

'Computers en de cybernetische samenleving' van M.A. Arbib

'De viewdata revolutie' van S. Fedida en R. Malik

MICROCOMPUTERS

'TRS-80 Basic: Een gids voor zelfstudie' van B. Albrecht e.a.

'CP/M: een gids voor zelfstudie' van J.N. Fernandez en R. Ashley

PROGRAMMEREN EN PROGRAMMEERTALEN

'Inleiding tot het programmeren, deel 1' van ir. J.J. van Amstel e.a.

'Inleiding tot het programmeren, deel 2' van ir. J.J. van Amstel e.a.

'Programmeren, deel 1: Inleiding' van prof.drs. C. Bron

'Programmeren, deel 2: Van analyse tot algoritme' van prof.drs. C. Bron

'Inleiding programmeren' van prof.dr. R.J. Lunbeck

'Inleiding datastructuren' van prof.dr. R.J. Lunbeck

'Inleiding programmeren en programmeertechnieken', EIT-serie, deel 1

'Het Groot Pascal Spreukenboek' van H.F. Ledgard, P.A. Nagin en J.F. Hueras

'Basic', EIT-serie, deel 3

'Cursus eenvoudig Pascal' van prof.dr. A. van der Sluis en drs. C.A.C. Görts

'Cursus Pascal' van prof.dr. A. van der Sluis en drs. C.A.C. Görts

'Inleiding programmeren in Pascal' van C. van de Wijgaart

'Cursus Fortran 77' van J.N.P. Hume en R.C. Holt

'Cursus COBOL' van A. Parkin

'Cursus Algol 60' van prof.dr. A. van der Sluis en drs. C.A.C. Görts

SYSTEEMPROGRAMMATUUR

'Computersystemen' van prof.ir. D.H. Wolbers

'Bedrijfssystemen', EIT-serie, deel 4

'Systeemprogrammatuur' van drs. H. Alblas

'Vertalerbouw' van drs. H. Alblas e.a.

DATASTRUCTUREN, BESTANDSORGANISATIE, DATABASE

'Bestandsorganisatie' van prof.dr. R.J. Lunbeck en drs. F. Remmen

'Gegevensstructuren' van R. Engmann e.a.

'Informatiestructuren, bestandsorganisatie en bestandsontwerp', EIT-serie, deel 5

INFORMATIEANALYSE, SYSTEEMONTWERP

'Eerlijk en helder', van prof.dr. P.G. Bosch

'Voorbereiding van computertoepassingen' van prof. A.B. Frielink

'Analyse van informatiebehoeften en de inhoudsbeschrijving van een databank' van prof.dr. P.G. Bosch en ir. H.M. Heemskerk

'IBSEN - Een SIMULA programma voor gebruik bij de beschrijving van informatiebehoeften' van prof.dr. P.G. Bosch en ir. H.M. Heemskerk

'Systeemontwikkeling volgens S.D.M.' van H.B. Eilers

'Een samenvatting van de System Development Methodology SDM' van PANDATA

'Gegevensanalyse' van R.P. Langerhorst

'Cases op het gebied van administratieve organisatie en informatie-verzorging (inclusief systeem-ontwerp)' van prof.dr. P.G. Bosch en H.A. Te Rijdt

'Uitwerkingenboek bij cases op het gebied van administratieve organisatie en informatieverwerking' van prof.dr. P.G. Bosch en H.A. Te Rijdt

'SIMULATIE, een moderne methode van onderzoek' van drs. S.K.T. Boersma en ir. T. Hoenderkamp

THEORETISCHE INFORMATICA

'Abstracte automaten en grammatica's' van prof.dr. A. Ollongren en ir. Th.P. van der Weide

AANVERWANTE ONDERWERPEN

'Lineaire programmering als hulpmiddel bij de besluitvorming' van drs. S.W. Douma

INFORMATIE OVER DEZE PUBLIKATIES BIJ:

Academic Service
Postbus 96996
2509 JJ 's-Gravenhage
(Tel. 070-247238)